Divide and Conquer

K. Subramani[1]

[1] Lane Department of Computer Science and Electrical Engineering
West Virginia University

1, 6 September 2016

Outline

Outline

Outline

Outline

Outline

Outline

Main Ideas

Main Ideas

Approach

Main Ideas

Approach

Main Ideas

Approach

1. **Divide** the problem into a number of subproblems that are smaller instances of the problem.

Main Ideas

Approach

1. **Divide** the problem into a number of subproblems that are smaller instances of the problem.
2. **Conquer** the subproblems by solving them recursively.

Main Ideas

Approach

1. **Divide** the problem into a number of subproblems that are smaller instances of the problem.
2. **Conquer** the subproblems by solving them recursively. If the subproblems are small enough, however, just solve the problems in a straightforward manner.

Main Ideas

Approach

1. **Divide** the problem into a number of subproblems that are smaller instances of the problem.
2. **Conquer** the subproblems by solving them recursively. If the subproblems are small enough, however, just solve the problems in a straightforward manner.
3. **Combine** the solutions to the subproblems into the solution for the original problem.

Main Ideas

Approach

1. **Divide** the problem into a number of subproblems that are smaller instances of the problem.
2. **Conquer** the subproblems by solving them recursively. If the subproblems are small enough, however, just solve the problems in a straightforward manner.
3. **Combine** the solutions to the subproblems into the solution for the original problem.

The Searching Problem

The Searching Problem

Statement of Problem

The Searching Problem

Statement of Problem

Given an array $\mathbf{A}[1 \cdot \cdot n]$ of integers, sorted in ascending order,

The Searching Problem

Statement of Problem

Given an array $\mathbf{A}[1 \cdot \cdot n]$ of integers, sorted in ascending order, and a key r,

The Searching Problem

Statement of Problem

Given an array $\mathbf{A}[1 \cdot \cdot n]$ of integers, sorted in ascending order, and a key r, check if $A[i] = r$, for any $1 \leq i \leq n$.

Linear Search

Linear Search

Algorithm

Linear Search

Algorithm

```
LINEAR-SEARCH(A, n, r)
if (n < 0) then
    return (false)
else
    if (A[n] = r) then
        return (true)
    else
        return (LINEAR-SEARCH(A, (n − 1), r))
    end if
end if
```

Linear Search

Algorithm

LINEAR-SEARCH(**A**, n, r)

if ($n < 0$) **then**
    return (**false**)
else
    if ($A[n] = r$) **then**
        return (**true**)
    else
        return (LINEAR-SEARCH(**A**, $(n-1)$, r))
    end if
end if

Analysis

Linear Search

Algorithm

```
LINEAR-SEARCH(A, n, r)
if (n < 0) then
    return (false)
else
    if (A[n] = r) then
        return (true)
    else
        return (LINEAR-SEARCH(A, (n - 1), r))
    end if
end if
```

Analysis

Prove correctness and establish bounds on number of element to element comparisons.

Binary Search

Binary Search

Algorithm

Binary Search

Algorithm

BINARY-SEARCH(**A**, *low*, *high*, *r*)

$\{$Initially $low = 1$, $high = n.\}$
if ($low \leq high$) **then**
    $mid = \frac{low+high}{2}$.
    if ($A[mid] = r$) **then**
        return (**true**)
    end if
    if ($A[mid] > r$) **then**
        return (BINARY-SEARCH(**A**, *low*, $mid - 1$, *r*))
    else
        return (BINARY-SEARCH(**A**, $mid + 1$, *high*, *r*))
    end if
else
    return (**false**)
end if

Binary Search (contd.)

Binary Search (contd.)

Analysis

Binary Search (contd.)

Analysis

Prove correctness and establish bounds on number of element to element comparisons.

The Sorting Problem

The Sorting Problem

Statement of Problem

The Sorting Problem

Statement of Problem

Given an array $\mathbf{A}[1 \cdot \cdot n]$ of integers, produce an ascending-order permutation of $\mathbf{A}$.

Merging Two Sorted Arrays

Merging Two Sorted Arrays

The Algorithm

Merging Two Sorted Arrays

The Algorithm

MERGE(**A**, *low*, *mid*, *high*)

$\{$We merge the arrays $\mathbf{A}[low \cdot\cdot mid]$ and $\mathbf{A}[mid+1 \cdot\cdot high]$.$\}$
Create a temporary array **B** of size $(mid - low + 1)$ and copy the elements from $\mathbf{A}[low \cdot\cdot mid]$ into this array.
Create a temporary array **C** of size $(high - mid)$ and copy the elements from $\mathbf{A}[mid+1 \cdot\cdot high]$ into this array.
Set $p = 1$, $q = 1$, $r = low$.
while $(p \leq (mid - low + 1)$ **and** $(q \leq (high - mid))$ **do**
    if $(B[p] \leq C[q])$ **then**
        $A[r] = B[p]$. $p++$.
    else
        $A[r] = C[q]$. $q++$.
    end if
    $r++$.
end while

Algorithm Merge (contd.)

Algorithm Merge (contd.)

The Algorithm (contd.)

MERGE(**A**, *low*, *mid*, *high*)

while $(p \leq (mid - low + 1))$ **do**
    $A[r] = B[p]$. $p++$. $r++$.
end while
while $(q \leq (high - mid))$ **do**
    $A[r] = C[q]$. $q++$. $r++$.
end while

Merge Sorting

Merge Sorting

The Algorithm

Merge Sorting

The Algorithm

MERGE-SORT(**A**, *low*, *high*)

1: $\{$Initially $low = 1$, $high = n.\}$
2: **if** ($low \leq high$) **then**
3: $\quad mid = \frac{low+high}{2}$.
4: $\quad$ MERGE-SORT(**A**, *low*, *mid*).
5: $\quad$ MERGE-SORT(**A**, $mid + 1$, *high*).
6: $\quad$ MERGE(**A**, *low*, *mid*, *high*).
7: **end if**

Merge Sorting

The Algorithm

$\textsc{Merge-Sort}(\mathbf{A}, low, high)$

```
{Initially low = 1, high = n.}
if (low ≤ high) then
    mid = (low+high)/2.
    MERGE-SORT(A, low, mid).
    MERGE-SORT(A, mid + 1, high).
    MERGE(A, low, mid, high).
end if
```

Analysis

Merge Sorting

The Algorithm

MERGE-SORT(**A**, *low*, *high*)

$\{$Initially $low = 1$, $high = n$.$\}$
if ($low \leq high$) **then**
    $mid = \frac{low+high}{2}$.
    MERGE-SORT(**A**, *low*, *mid*).
    MERGE-SORT(**A**, $mid + 1$, *high*).
    MERGE(**A**, *low*, *mid*, *high*).
end if

Analysis

Prove correctness and establish bounds on number of element to element comparisons.

The Partition subroutine

The Partition subroutine

Partitioning

The Partition subroutine

Partitioning

Function PARTITION($\mathbf{A}, p, q$)

$\{$We partition the sub-array $\mathbf{A}[p, p+1, \ldots, q]$ about $A[p]$.$\}$
for $(i = (p+1)$ **to**; $q)$ **do**
  if $(A[i] < A[p])$ **then**
    Insert $A[i]$ into bucket $\mathbf{L}$.
  else
    if $(A[i] > A[p])$ **then**
      Insert $A[i]$ into bucket $\mathbf{U}$.
    end if
  end if
end for
Copy $A[p]$ into $A[(|\mathbf{L}| + 1)]$.
Copy the elements of $\mathbf{L}$ into the first $|\mathbf{L}|$ entries of $\mathbf{A}[p \cdot \cdot q]$.
Copy $A[p]$ into $A[(|\mathbf{L}| + 1)]$.
Copy the elements of $\mathbf{U}$ into the entries of $\mathbf{A}[(|\mathbf{L}| + 2) \cdot \cdot q]$.
return $(|\mathbf{L}| + 1)$.

The Partition subroutine

Partitioning

Function PARTITION($\mathbf{A}, p, q$)

$\{$We partition the sub-array $\mathbf{A}[p, p+1, \ldots, q]$ about $A[p]$.$\}$
for $(i = (p+1)$ **to**; $q)$ **do**
    if $(A[i] < A[p])$ **then**
        Insert $A[i]$ into bucket $\mathbf{L}$.
    else
        if $(A[i] > A[p])$ **then**
            Insert $A[i]$ into bucket $\mathbf{U}$.
        end if
    end if
end for
Copy $A[p]$ into $A[(|\mathbf{L}| + 1)]$.
Copy the elements of $\mathbf{L}$ into the first $|\mathbf{L}|$ entries of $\mathbf{A}[p \cdot \cdot q]$.
Copy $A[p]$ into $A[(|\mathbf{L}| + 1)]$.
Copy the elements of $\mathbf{U}$ into the entries of $\mathbf{A}[(|\mathbf{L}| + 2) \cdot \cdot q]$.
return $(|\mathbf{L}| + 1)$.

The Quick-sort Algorithm

The Quick-sort Algorithm

Note

The Quick-sort Algorithm

Note

Partitioning an array can be achieved in linear time.

The Quick-sort Algorithm

Note

Partitioning an array can be achieved in linear time.

The Algorithm

The Quick-sort Algorithm

Note

Partitioning an array can be achieved in linear time.

The Algorithm

Function QUICK-SORT($\mathbf{A}$, p, q)

1: **if** $(p \geq q)$ **then**
2:     **return**
3: **else**
4:     $j =$PARTITION($\mathbf{A}$, p, q).
5:     Quicksort($\mathbf{A}$, p, $j-1$).
6:     Quicksort($\mathbf{A}$, $j+1$, q).
7: **end if**

The Quick-sort Algorithm

Note

Partitioning an array can be achieved in linear time.

The Algorithm

Function QUICK-SORT(**A**, p, q)

1: **if** ($p \geq q$) **then**
2:     **return**
3: **else**
4:     $j =$PARTITION(**A**, p, q).
5:     Quicksort(**A**, p, $j - 1$).
6:     Quicksort(**A**, $j + 1$, q).
7: **end if**

Note

The main program calls QUICK-SORT(**A**, 1, n).

Worst-case analysis

Analysis

What is the worst-case input for QUICKSORT()?

Worst-case analysis

Analysis

What is the worst-case input for QUICKSORT()? How many comparisons in the worst case?

Worst-case analysis

Analysis

What is the worst-case input for QUICKSORT()? How many comparisons in the worst case? $O(n^2)$.

Worst-case analysis

Analysis

What is the worst-case input for QUICKSORT()? How many comparisons in the worst case? $O(n^2)$.

Intuition for randomized case

What sort of assumptions are reasonable in analysis?

Randomized Quicksort

Randomized Quicksort

The Algorithm

Randomized Quicksort

The Algorithm

Function RANDOMIZED-QUICKSORT(**A**, p, q)

if $(p \geq q)$ **then**
    return
else
    Choose a number, say r, uniformly and at random from the set $\{p, p+1, \ldots, q\}$.
    Swap $A[p]$ and $A[r]$.
    $j =$PARTITION(**A**, p, q).
    Quicksort(**A**, p, $j-1$).
    Quicksort(**A**, $j+1$, q).
end if

Randomized Quicksort

The Algorithm

Function RANDOMIZED-QUICKSORT(**A**, p, q)

```
if (p ≥ q) then
    return
else
    Choose a number, say r, uniformly and at random from the set {p, p + 1, ..., q}.
    Swap A[p] and A[r].
    j = PARTITION(A, p, q).
    Quicksort(A, p, j − 1).
    Quicksort(A, j + 1, q).
end if
```

Note

Randomized Quicksort

The Algorithm

Function RANDOMIZED-QUICKSORT(**A**, p, q)

if $(p \geq q)$ **then**
    return
else
    Choose a number, say r, uniformly and at random from the set $\{p, p+1, \ldots, q\}$.
    Swap $A[p]$ and $A[r]$.
    $j =$ PARTITION(**A**, p, q).
    Quicksort(**A**, p, $j-1$).
    Quicksort(**A**, $j+1$, q).
end if

Note

Worst case running time?

Randomized Quicksort

The Algorithm

Function RANDOMIZED-QUICKSORT($\mathbf{A},\ p,\ q$)

if ($p \geq q$) **then**
    return
else
    Choose a number, say r, uniformly and at random from the set $\{p, p+1, \ldots, q\}$.
    Swap $A[p]$ and $A[r]$.
    $j =$PARTITION($\mathbf{A},\ p,\ q$).
    Quicksort($\mathbf{A},\ p,\ j-1$).
    Quicksort($\mathbf{A},\ j+1,\ q$).
end if

Note

Worst case running time? $O(n^2)$*!*

Randomized Quicksort

The Algorithm

Function RANDOMIZED-QUICKSORT(**A**, p, q)

```
if (p ≥ q) then
    return
else
    Choose a number, say r, uniformly and at random from the set {p, p + 1, ..., q}.
    Swap A[p] and A[r].
    j = PARTITION(A, p, q).
    Quicksort(A, p, j − 1).
    Quicksort(A, j + 1, q).
end if
```

Note

Worst case running time? $O(n^2)$*! However, for a randomized algorithm we are not interested in worst-case running time, but in expected running time.*

Decision Tree Analysis

Decision Tree Analysis

Decision Tree

Decision Tree Analysis

Decision Tree

The operation of RANDOMIZED QUICKSORT() can be thought of as a binary tree, say T, with a pivot being chosen at each internal node.

Decision Tree Analysis

Decision Tree

The operation of RANDOMIZED QUICKSORT() can be thought of as a binary tree, say T, with a pivot being chosen at each internal node.

The elements in the node which are less than the pivot are shunted to the left subtree and the rest of the elements (excluding the pivot) are shunted to the right subtree.

Decision Tree Analysis

Decision Tree

The operation of RANDOMIZED QUICKSORT() can be thought of as a binary tree, say T, with a pivot being chosen at each internal node.

The elements in the node which are less than the pivot are shunted to the left subtree and the rest of the elements (excluding the pivot) are shunted to the right subtree.

An in-order traversal of T focusing on the pivots, gives the sorted order.

Decision Tree Analysis

Decision Tree

The operation of RANDOMIZED QUICKSORT() can be thought of as a binary tree, say T, with a pivot being chosen at each internal node.

The elements in the node which are less than the pivot are shunted to the left subtree and the rest of the elements (excluding the pivot) are shunted to the right subtree.

An in-order traversal of T focusing on the pivots, gives the sorted order.

What is the work done at each level of the tree?

Decision Tree Analysis

Decision Tree

The operation of RANDOMIZED QUICKSORT() can be thought of as a binary tree, say T, with a pivot being chosen at each internal node.

The elements in the node which are less than the pivot are shunted to the left subtree and the rest of the elements (excluding the pivot) are shunted to the right subtree.

An in-order traversal of T focusing on the pivots, gives the sorted order.

What is the work done at each level of the tree? $O(n)$.

Decision Tree Analysis

Decision Tree

The operation of RANDOMIZED QUICKSORT() can be thought of as a binary tree, say T, with a pivot being chosen at each internal node.

The elements in the node which are less than the pivot are shunted to the left subtree and the rest of the elements (excluding the pivot) are shunted to the right subtree.

An in-order traversal of T focusing on the pivots, gives the sorted order.

What is the work done at each level of the tree? $O(n)$.

Let h denote the height of T.

Decision Tree Analysis

Decision Tree

The operation of RANDOMIZED QUICKSORT() can be thought of as a binary tree, say T, with a pivot being chosen at each internal node.

The elements in the node which are less than the pivot are shunted to the left subtree and the rest of the elements (excluding the pivot) are shunted to the right subtree.

An in-order traversal of T focusing on the pivots, gives the sorted order.

What is the work done at each level of the tree? $O(n)$.

Let h denote the height of T. Observe that h is a random variable and we are interested in its expected value.

Decision Tree Analysis

Decision Tree

The operation of RANDOMIZED QUICKSORT() can be thought of as a binary tree, say T, with a pivot being chosen at each internal node.

The elements in the node which are less than the pivot are shunted to the left subtree and the rest of the elements (excluding the pivot) are shunted to the right subtree.

An in-order traversal of T focusing on the pivots, gives the sorted order.

What is the work done at each level of the tree? $O(n)$.

Let h denote the height of T. Observe that h is a random variable and we are interested in its expected value.

The rank of an element of $\mathbf{A}$ is its position in $\mathbf{A}$, when $\mathbf{A}$ has been sorted.

Decision Tree Analysis

Decision Tree

The operation of RANDOMIZED QUICKSORT() can be thought of as a binary tree, say T, with a pivot being chosen at each internal node.

The elements in the node which are less than the pivot are shunted to the left subtree and the rest of the elements (excluding the pivot) are shunted to the right subtree.

An in-order traversal of T focusing on the pivots, gives the sorted order.

What is the work done at each level of the tree? $O(n)$.

Let h denote the height of T. Observe that h is a random variable and we are interested in its expected value.

The rank of an element of **A** is its position in **A**, when **A** has been sorted.

When you pick an element at random, what is the probability that the rank of the element chosen is between $\frac{1}{4} \cdot n$ and $\frac{3}{4} \cdot n$, where n is the number of elements in the array?

Decision Tree Analysis

Decision Tree

The operation of RANDOMIZED QUICKSORT() can be thought of as a binary tree, say T, with a pivot being chosen at each internal node.

The elements in the node which are less than the pivot are shunted to the left subtree and the rest of the elements (excluding the pivot) are shunted to the right subtree.

An in-order traversal of T focusing on the pivots, gives the sorted order.

What is the work done at each level of the tree? $O(n)$.

Let h denote the height of T. Observe that h is a random variable and we are interested in its expected value.

The rank of an element of $\mathbf{A}$ is its position in $\mathbf{A}$, when $\mathbf{A}$ has been sorted.

When you pick an element at random, what is the probability that the rank of the element chosen is between $\frac{1}{4} \cdot n$ and $\frac{3}{4} \cdot n$, where n is the number of elements in the array? $\frac{1}{2}$.

Decision Tree Analysis (contd.)

Decision Tree Analysis (contd.)

Analysis

Decision Tree Analysis (contd.)

Analysis

Consider the tree T.

Decision Tree Analysis (contd.)

Analysis

Consider the tree T.

We define an internal node o of the tree to be **good**, if both its children have at most $\frac{3}{4} \cdot |o|$ nodes, where $|o|$ denotes the number of elements in the node o.

Decision Tree Analysis (contd.)

Analysis

Consider the tree T.

We define an internal node o of the tree to be **good**, if both its children have at most $\frac{3}{4} \cdot |o|$ nodes, where $|o|$ denotes the number of elements in the node o.

Given an internal node o of T, what is the probability that it is **good**?

Decision Tree Analysis (contd.)

Analysis

Consider the tree T.

We define an internal node o of the tree to be **good**, if both its children have at most $\frac{3}{4} \cdot |o|$ nodes, where $|o|$ denotes the number of elements in the node o.

Given an internal node o of T, what is the probability that it is **good**? At least $\frac{1}{2}$!

Decision Tree Analysis (contd.)

Analysis

Consider the tree T.

We define an internal node o of the tree to be **good**, if both its children have at most $\frac{3}{4} \cdot |o|$ nodes, where $|o|$ denotes the number of elements in the node o.

Given an internal node o of T, what is the probability that it is **good**? At least $\frac{1}{2}$!

Consider a root to leaf path in T.

Decision Tree Analysis (contd.)

Analysis

Consider the tree T.

We define an internal node o of the tree to be **good**, if both its children have at most $\frac{3}{4} \cdot |o|$ nodes, where $|o|$ denotes the number of elements in the node o.

Given an internal node o of T, what is the probability that it is **good**? At least $\frac{1}{2}$!

Consider a root to leaf path in T.

How many **good** nodes can exist on such a path?

Decision Tree Analysis (contd.)

Analysis

Consider the tree T.

We define an internal node o of the tree to be **good**, if both its children have at most $\frac{3}{4} \cdot |o|$ nodes, where $|o|$ denotes the number of elements in the node o.

Given an internal node o of T, what is the probability that it is **good**? At least $\frac{1}{2}$!

Consider a root to leaf path in T.

How many **good** nodes can exist on such a path?

At most $r = \log_{\frac{4}{3}} n$.

Decision Tree Analysis (contd.)

Analysis

Consider the tree T.

We define an internal node o of the tree to be **good**, if both its children have at most $\frac{3}{4} \cdot |o|$ nodes, where $|o|$ denotes the number of elements in the node o.

Given an internal node o of T, what is the probability that it is **good**? At least $\frac{1}{2}$!

Consider a root to leaf path in T.

How many **good** nodes can exist on such a path?

At most $r = \log_{\frac{4}{3}} n$.

What is the *expected* number of nodes on a root to leaf path before you see r good nodes?

Decision Tree Analysis (contd.)

Lemma

Consider a coin for which the probability of "heads" turning up on a toss is p.

Decision Tree Analysis (contd.)

Lemma

Consider a coin for which the probability of "heads" turning up on a toss is p. What is the expected number of tosses to obtain k heads?

Decision Tree Analysis (contd.)

Lemma

Consider a coin for which the probability of "heads" turning up on a toss is p. What is the expected number of tosses to obtain k heads? $\frac{k}{p}$.

Decision Tree (contd.)

Decision Tree Analysis (contd.)

Lemma

Consider a coin for which the probability of "heads" turning up on a toss is p. What is the expected number of tosses to obtain k heads? $\frac{k}{p}$.

Decision Tree (contd.)

Thus the expected number of nodes on a root to leaf path is

Decision Tree Analysis (contd.)

Lemma

Consider a coin for which the probability of "heads" turning up on a toss is p. What is the expected number of tosses to obtain k heads? $\frac{k}{p}$.

Decision Tree (contd.)

Thus the expected number of nodes on a root to leaf path is $\frac{r}{\frac{1}{2}} =$

Decision Tree Analysis (contd.)

Lemma

Consider a coin for which the probability of "heads" turning up on a toss is p. What is the expected number of tosses to obtain k heads? $\frac{k}{p}$.

Decision Tree (contd.)

Thus the expected number of nodes on a root to leaf path is $\frac{r}{\frac{1}{2}} = 2 \cdot r =$

Decision Tree Analysis (contd.)

Lemma

Consider a coin for which the probability of "heads" turning up on a toss is p. What is the expected number of tosses to obtain k heads? $\frac{k}{p}$.

Decision Tree (contd.)

Thus the expected number of nodes on a root to leaf path is $\frac{r}{\frac{1}{2}} = 2 \cdot r = 2 \cdot \log_{\frac{4}{3}} n$.

Decision Tree Analysis (contd.)

Lemma

Consider a coin for which the probability of "heads" turning up on a toss is p. What is the expected number of tosses to obtain k heads? $\frac{k}{p}$.

Decision Tree (contd.)

Thus the expected number of nodes on a root to leaf path is $\frac{r}{\frac{1}{2}} = 2 \cdot r = 2 \cdot \log_{\frac{4}{3}} n$.

However, this is the expected height of T, i.e., $E[h]$.

Decision Tree Analysis (contd.)

Lemma

Consider a coin for which the probability of "heads" turning up on a toss is p. What is the expected number of tosses to obtain k heads? $\frac{k}{p}$.

Decision Tree (contd.)

Thus the expected number of nodes on a root to leaf path is $\frac{r}{\frac{1}{2}} = 2 \cdot r = 2 \cdot \log_{\frac{4}{3}} n$.

However, this is the expected height of T, i.e., $E[h]$.

Therefore, the expected work undertaken by the algorithm

Decision Tree Analysis (contd.)

Lemma

Consider a coin for which the probability of "heads" turning up on a toss is p. What is the expected number of tosses to obtain k heads? $\frac{k}{p}$.

Decision Tree (contd.)

Thus the expected number of nodes on a root to leaf path is $\frac{r}{\frac{1}{2}} = 2 \cdot r = 2 \cdot \log_{\frac{4}{3}} n$.

However, this is the expected height of T, i.e., $E[h]$.

Therefore, the expected work undertaken by the algorithm

$$E[h] \times \text{work done per level} \quad = \quad O(n \cdot \log n).$$

Indicator Variable Analysis

Indicator Variable Analysis

Definition

Indicator Variable Analysis

Definition

A random variable is an indicator variable, if it assumes the value 1, for the occurrence of some event,

Indicator Variable Analysis

Definition

A random variable is an indicator variable, if it assumes the value 1, for the occurrence of some event, and 0 otherwise.

Indicator Variable Analysis

Definition

A random variable is an indicator variable, if it assumes the value 1, for the occurrence of some event, and 0 otherwise.

Note

Indicator Variable Analysis

Definition

A random variable is an indicator variable, if it assumes the value 1, for the occurrence of some event, and 0 otherwise.

Note

We recall that the rank of an array element is its position in the sorted array.

Indicator Variable Analysis

Definition

A random variable is an indicator variable, if it assumes the value 1, for the occurrence of some event, and 0 otherwise.

Note

We recall that the rank of an array element is its position in the sorted array. Every element of $\mathbf{A}$ *has a unique rank in the set* $\{1, 2, \ldots, n\}$*.*

Indicator Variable Analysis

Definition

A random variable is an indicator variable, if it assumes the value 1, for the occurrence of some event, and 0 otherwise.

Note

We recall that the rank of an array element is its position in the sorted array. Every element of $\mathbf{A}$ *has a unique rank in the set* $\{1, 2, \ldots, n\}$.

Analysis

Indicator Variable Analysis

Definition

A random variable is an indicator variable, if it assumes the value 1, for the occurrence of some event, and 0 otherwise.

Note

We recall that the rank of an array element is its position in the sorted array. Every element of **A** *has a unique rank in the set* $\{1, 2, \ldots, n\}$*.*

Analysis

Let $S(i)$ denote the element in **A**, whose rank is i.

Indicator Variable Analysis

Definition

A random variable is an indicator variable, if it assumes the value 1, for the occurrence of some event, and 0 otherwise.

Note

We recall that the rank of an array element is its position in the sorted array. Every element of **A** *has a unique rank in the set* $\{1, 2, \ldots, n\}$.

Analysis

Let $S(i)$ denote the element in **A**, whose rank is i.

We wish to compute the number of comparisons between $A[i]$ and the other elements of **A**, for each $i = 1, 2 \ldots n$.

Indicator Variable Analysis

Definition

A random variable is an indicator variable, if it assumes the value 1, for the occurrence of some event, and 0 otherwise.

Note

We recall that the rank of an array element is its position in the sorted array. Every element of **A** *has a unique rank in the set* $\{1, 2, \ldots, n\}$.

Analysis

Let $S(i)$ denote the element in **A**, whose rank is i.

We wish to compute the number of comparisons between $A[i]$ and the other elements of **A**, for each $i = 1, 2 \ldots n$.

Instead, we will compute the number of comparisons between $S(i)$ and the elements of other ranks, for each $i = 1, 2, \ldots, n$.

Indicator Variable Analysis

Definition

A random variable is an indicator variable, if it assumes the value 1, for the occurrence of some event, and 0 otherwise.

Note

We recall that the rank of an array element is its position in the sorted array. Every element of **A** *has a unique rank in the set* $\{1, 2, \ldots, n\}$.

Analysis

Let $S(i)$ denote the element in **A**, whose rank is i.

We wish to compute the number of comparisons between $A[i]$ and the other elements of **A**, for each $i = 1, 2 \ldots n$.

Instead, we will compute the number of comparisons between $S(i)$ and the elements of other ranks, for each $i = 1, 2, \ldots, n$.

Are the two computations equivalent?

Indicator Variable Analysis (contd.)

Indicator Variable Analysis (contd.)

Analysis (contd.)

Indicator Variable Analysis (contd.)

Analysis (contd.)

Let X_{ij} denote an indicator random variable, defined as follows:

Indicator Variable Analysis (contd.)

Analysis (contd.)

Let X_{ij} denote an indicator random variable, defined as follows:

$$X_{ij} = \begin{cases} 1, & \text{if S(i) and S(j) are compared during the course of the algorithm} \\ 0, & \text{otherwise} \end{cases}$$

Indicator Variable Analysis (contd.)

Analysis (contd.)

Let X_{ij} denote an indicator random variable, defined as follows:

$$X_{ij} = \begin{cases} 1, & \text{if S(i) and S(j) are compared during the course of the algorithm} \\ 0, & \text{otherwise} \end{cases}$$

Let X denote the total number of comparisons made by the algorithm.

Indicator Variable Analysis (contd.)

Analysis (contd.)

Let X_{ij} denote an indicator random variable, defined as follows:

$$X_{ij} = \begin{cases} 1, & \text{if S(i) and S(j) are compared during the course of the algorithm} \\ 0, & \text{otherwise} \end{cases}$$

Let X denote the total number of comparisons made by the algorithm. Clearly,

Indicator Variable Analysis (contd.)

Analysis (contd.)

Let X_{ij} denote an indicator random variable, defined as follows:

$$X_{ij} = \begin{cases} 1, & \text{if S(i) and S(j) are compared during the course of the algorithm} \\ 0, & \text{otherwise} \end{cases}$$

Let X denote the total number of comparisons made by the algorithm. Clearly,

$$X =$$

Indicator Variable Analysis (contd.)

Analysis (contd.)

Let X_{ij} denote an indicator random variable, defined as follows:

$$X_{ij} = \begin{cases} 1, & \text{if S(i) and S(j) are compared during the course of the algorithm} \\ 0, & \text{otherwise} \end{cases}$$

Let X denote the total number of comparisons made by the algorithm. Clearly,

$$X = \sum_{i=1}^{n-1} \sum_{j>i} X_{ij}$$

Indicator Variable Analysis (contd.)

Analysis (contd.)

Let X_{ij} denote an indicator random variable, defined as follows:

$$X_{ij} = \begin{cases} 1, & \text{if S(i) and S(j) are compared during the course of the algorithm} \\ 0, & \text{otherwise} \end{cases}$$

Let X denote the total number of comparisons made by the algorithm. Clearly,

$$\begin{aligned} X &= \sum_{i=1}^{n-1} \sum_{j>i} X_{ij} \\ &= \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} X_{ij} \end{aligned}$$

Indicator Variable Analysis (contd.)

Analysis (contd.)

Let X_{ij} denote an indicator random variable, defined as follows:

$$X_{ij} = \begin{cases} 1, & \text{if S(i) and S(j) are compared during the course of the algorithm} \\ 0, & \text{otherwise} \end{cases}$$

Let X denote the total number of comparisons made by the algorithm. Clearly,

$$\begin{aligned} X &= \sum_{i=1}^{n-1} \sum_{j>i} X_{ij} \\ &= \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} X_{ij} \end{aligned}$$

How to compute X?

Indicator Variable Analysis (contd.)

Analysis (contd.)

Let X_{ij} denote an indicator random variable, defined as follows:

$$X_{ij} = \begin{cases} 1, & \text{if S(i) and S(j) are compared during the course of the algorithm} \\ 0, & \text{otherwise} \end{cases}$$

Let X denote the total number of comparisons made by the algorithm. Clearly,

$$\begin{aligned} X &= \sum_{i=1}^{n-1} \sum_{j>i} X_{ij} \\ &= \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} X_{ij} \end{aligned}$$

How to compute X? We are not interested in X, but in $E[X]$!

Indicator Variable Analysis (contd.)

Analysis (contd.)

Let X_{ij} denote an indicator random variable, defined as follows:

$$X_{ij} = \begin{cases} 1, & \text{if S(i) and S(j) are compared during the course of the algorithm} \\ 0, & \text{otherwise} \end{cases}$$

Let X denote the total number of comparisons made by the algorithm. Clearly,

$$\begin{aligned} X &= \sum_{i=1}^{n-1} \sum_{j>i} X_{ij} \\ &= \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} X_{ij} \end{aligned}$$

How to compute X? We are not interested in X, but in $E[X]$!

Indicator Variable Analysis (contd.)

Indicator Variable Analysis (contd.)

Analysis (contd.)

Indicator Variable Analysis (contd.)

Analysis (contd.)

Observe that,

$$E[X] = E[\sum_{i=1}^{n-1} \sum_{j=i+1}^{n} X_{ij}]$$

Indicator Variable Analysis (contd.)

Analysis (contd.)

Observe that,

$$
\begin{aligned}
E[X] &= E[\sum_{i=1}^{n-1} \sum_{j=i+1}^{n} X_{ij}] \\
&=
\end{aligned}
$$

Indicator Variable Analysis (contd.)

Analysis (contd.)

Observe that,

$$\begin{aligned} E[X] &= E[\sum_{i=1}^{n-1} \sum_{j=i+1}^{n} X_{ij}] \\ &= \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} E[X_{ij}] \end{aligned}$$

Indicator Variable Analysis (contd.)

Analysis (contd.)

Observe that,

$$\begin{aligned} E[X] &= E[\sum_{i=1}^{n-1}\sum_{j=i+1}^{n} X_{ij}] \\ &= \sum_{i=1}^{n-1}\sum_{j=i+1}^{n} E[X_{ij}] \end{aligned}$$

Let p_{ij} denote the probability that $S(i)$ and $S(j)$ are compared.

Indicator Variable Analysis (contd.)

Analysis (contd.)

Observe that,

$$E[X] = E[\sum_{i=1}^{n-1} \sum_{j=i+1}^{n} X_{ij}]$$
$$= \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} E[X_{ij}]$$

Let p_{ij} denote the probability that $S(i)$ and $S(j)$ are compared.

Clearly, $E[X_{ij}] =$

Indicator Variable Analysis (contd.)

Analysis (contd.)

Observe that,

$$\begin{aligned} E[X] &= E[\sum_{i=1}^{n-1} \sum_{j=i+1}^{n} X_{ij}] \\ &= \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} E[X_{ij}] \end{aligned}$$

Let p_{ij} denote the probability that $S(i)$ and $S(j)$ are compared.

Clearly, $E[X_{ij}] = p_{ij}$.

Indicator Variable Analysis (contd.)

Analysis (contd.)

Observe that,

$$
\begin{aligned}
E[X] &= E[\sum_{i=1}^{n-1} \sum_{j=i+1}^{n} X_{ij}] \\
&= \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} E[X_{ij}]
\end{aligned}
$$

Let p_{ij} denote the probability that $S(i)$ and $S(j)$ are compared.

Clearly, $E[X_{ij}] = p_{ij}$.

How to compute p_{ij}?

Indicator Variable Analysis (contd.)

Analysis (contd.)

Observe that,

$$
\begin{aligned}
E[X] &= E[\sum_{i=1}^{n-1} \sum_{j=i+1}^{n} X_{ij}] \\
&= \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} E[X_{ij}]
\end{aligned}
$$

Let p_{ij} denote the probability that $S(i)$ and $S(j)$ are compared.

Clearly, $E[X_{ij}] = p_{ij}$.

How to compute p_{ij}?

Let $S_{ij} = \{S(i), S(i+1), \ldots S(j)\}$.

Indicator Variable Analysis (contd.)

Analysis (contd.)

Observe that,

$$E[X] = E[\sum_{i=1}^{n-1} \sum_{j=i+1}^{n} X_{ij}]$$
$$= \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} E[X_{ij}]$$

Let p_{ij} denote the probability that $S(i)$ and $S(j)$ are compared.

Clearly, $E[X_{ij}] = p_{ij}$.

How to compute p_{ij}?

Let $S_{ij} = \{S(i), S(i+1), \ldots S(j)\}$. $S(i)$ and $S(j)$ will be compared **only if**, either one of them is picked before the other elements in S_{ij}!

Indicator Variable Analysis (contd.)

Analysis (contd.)

Observe that,

$$\begin{aligned} E[X] &= E[\sum_{i=1}^{n-1} \sum_{j=i+1}^{n} X_{ij}] \\ &= \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} E[X_{ij}] \end{aligned}$$

Let p_{ij} denote the probability that $S(i)$ and $S(j)$ are compared.

Clearly, $E[X_{ij}] = p_{ij}$.

How to compute p_{ij}?

Let $S_{ij} = \{S(i), S(i+1), \ldots S(j)\}$. $S(i)$ and $S(j)$ will be compared **only if**, either one of them is picked before the other elements in S_{ij}!

Since all choices are made uniformly and at random, the probability of either $S(i)$ or $S(j)$ being picked before the other elements in S_{ij} is exactly

Indicator Variable Analysis (contd.)

Analysis (contd.)

Observe that,

$$\begin{aligned} E[X] &= E[\sum_{i=1}^{n-1} \sum_{j=i+1}^{n} X_{ij}] \\ &= \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} E[X_{ij}] \end{aligned}$$

Let p_{ij} denote the probability that $S(i)$ and $S(j)$ are compared.

Clearly, $E[X_{ij}] = p_{ij}$.

How to compute p_{ij}?

Let $S_{ij} = \{S(i), S(i+1), \ldots S(j)\}$. $S(i)$ and $S(j)$ will be compared **only if**, either one of them is picked before the other elements in S_{ij}!

Since all choices are made uniformly and at random, the probability of either $S(i)$ or $S(j)$ being picked before the other elements in S_{ij} is exactly $\frac{2}{j-i+1}$.

Indicator Variable Analysis (contd.)

Analysis (contd.)

Observe that,

$$\begin{aligned} E[X] &= E[\sum_{i=1}^{n-1} \sum_{j=i+1}^{n} X_{ij}] \\ &= \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} E[X_{ij}] \end{aligned}$$

Let p_{ij} denote the probability that $S(i)$ and $S(j)$ are compared.

Clearly, $E[X_{ij}] = p_{ij}$.

How to compute p_{ij}?

Let $S_{ij} = \{S(i), S(i+1), \ldots S(j)\}$. $S(i)$ and $S(j)$ will be compared **only if**, either one of them is picked before the other elements in S_{ij}!

Since all choices are made uniformly and at random, the probability of either $S(i)$ or $S(j)$ being picked before the other elements in S_{ij} is exactly $\frac{2}{j-i+1}$.

Indicator Variable Analysis (contd.)

Indicator Variable Analysis (contd.)

Analysis (contd.)

Indicator Variable Analysis (contd.)

Analysis (contd.)

Therefore,

Indicator Variable Analysis (contd.)

Analysis (contd.)

Therefore,

$$E[X] = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} E[X_{ij}]$$

Indicator Variable Analysis (contd.)

Analysis (contd.)

Therefore,

$$\begin{aligned} E[X] &= \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} E[X_{ij}] \\ &= \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \frac{2}{j-i+1} \end{aligned}$$

Indicator Variable Analysis (contd.)

Analysis (contd.)

Therefore,

$$
\begin{aligned}
E[X] &= \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} E[X_{ij}] \\
&= \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \frac{2}{j-i+1} \\
&= \sum_{i=1}^{n-1} \sum_{k=2}^{n-i+1} \frac{2}{k}
\end{aligned}
$$

Indicator Variable Analysis (contd.)

Analysis (contd.)

Therefore,

$$\begin{aligned}
E[X] &= \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} E[X_{ij}] \\
&= \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \frac{2}{j-i+1} \\
&= \sum_{i=1}^{n-1} \sum_{k=2}^{n-i+1} \frac{2}{k} \\
&\leq \sum_{i=1}^{n} \sum_{k=1}^{n} \frac{2}{k}
\end{aligned}$$

Indicator Variable Analysis (contd.)

Analysis (contd.)

Therefore,

$$\begin{aligned} E[X] &= \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} E[X_{ij}] \\ &= \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \frac{2}{j-i+1} \\ &= \sum_{i=1}^{n-1} \sum_{k=2}^{n-i+1} \frac{2}{k} \\ &\leq \sum_{i=1}^{n} \sum_{k=1}^{n} \frac{2}{k} \end{aligned}$$

Final Steps

Final Steps

Concluding the analysis

Final Steps

Concluding the analysis

$$E[X] \leq$$

Final Steps

Concluding the analysis

$$E[X] \quad \leq \quad 2 \cdot \sum_{i=1}^{n} \sum_{k=1}^{n} \frac{1}{k}$$

Final Steps

Concluding the analysis

$$
\begin{aligned}
E[X] &\leq 2 \cdot \sum_{i=1}^{n} \sum_{k=1}^{n} \frac{1}{k} \\
&= 2 \cdot \sum_{i=1}^{n} H_n
\end{aligned}
$$

Final Steps

Concluding the analysis

$$
\begin{aligned}
E[X] &\leq 2 \cdot \sum_{i=1}^{n} \sum_{k=1}^{n} \frac{1}{k} \\
&= 2 \cdot \sum_{i=1}^{n} H_n \\
&= 2 \cdot n \cdot H_n
\end{aligned}
$$

Final Steps

Concluding the analysis

$$
\begin{aligned}
E[X] &\leq 2 \cdot \sum_{i=1}^{n} \sum_{k=1}^{n} \frac{1}{k} \\
&= 2 \cdot \sum_{i=1}^{n} H_n \\
&= 2 \cdot n \cdot H_n \\
&\in O(n \cdot \log n)
\end{aligned}
$$

Integer Multiplication

Integer Multiplication

Main Issues

Integer Multiplication

Main Issues

Integer Multiplication

Main Issues

1. Two n-bit numbers I and J can be added and subtracted in $O(n)$ time.

Integer Multiplication

Main Issues

1. Two n-bit numbers I and J can be added and subtracted in $O(n)$ time.
2. Standard multiplication takes $O(n^2)$ time.

Integer Multiplication

Main Issues

1. Two n-bit numbers I and J can be added and subtracted in $O(n)$ time.
2. Standard multiplication takes $O(n^2)$ time. (Why?)

Integer Multiplication

Main Issues

1. Two n-bit numbers I and J can be added and subtracted in $O(n)$ time.
2. Standard multiplication takes $O(n^2)$ time. (Why?)
3. Can we do better?

Integer Multiplication

Main Issues

1. Two n-bit numbers I and J can be added and subtracted in $O(n)$ time.
2. Standard multiplication takes $O(n^2)$ time. (Why?)
3. Can we do better? Observe that,

Integer Multiplication

Main Issues

1. Two n-bit numbers I and J can be added and subtracted in $O(n)$ time.
2. Standard multiplication takes $O(n^2)$ time. (Why?)
3. Can we do better? Observe that,

$$I = I_h \cdot 2^{\frac{n}{2}} + I_l$$

Integer Multiplication

Main Issues

1. Two n-bit numbers I and J can be added and subtracted in $O(n)$ time.
2. Standard multiplication takes $O(n^2)$ time. (Why?)
3. Can we do better? Observe that,

$$\begin{aligned} I &= I_h \cdot 2^{\frac{n}{2}} + I_l \\ J &= J_h \cdot 2^{\frac{n}{2}} + J_l \end{aligned}$$

Integer Multiplication

Main Issues

1. Two n-bit numbers I and J can be added and subtracted in $O(n)$ time.
2. Standard multiplication takes $O(n^2)$ time. (Why?)
3. Can we do better? Observe that,

$$\begin{aligned} I &= I_h \cdot 2^{\frac{n}{2}} + I_l \\ J &= J_h \cdot 2^{\frac{n}{2}} + J_l \end{aligned}$$

4. It follows that,

Integer Multiplication

Main Issues

1. Two n-bit numbers I and J can be added and subtracted in $O(n)$ time.
2. Standard multiplication takes $O(n^2)$ time. (Why?)
3. Can we do better? Observe that,

$$\begin{aligned} I &= I_h \cdot 2^{\frac{n}{2}} + I_l \\ J &= J_h \cdot 2^{\frac{n}{2}} + J_l \end{aligned}$$

4. It follows that,

$$I \cdot J =$$

Integer Multiplication

Main Issues

1. Two n-bit numbers I and J can be added and subtracted in $O(n)$ time.
2. Standard multiplication takes $O(n^2)$ time. (Why?)
3. Can we do better? Observe that,

$$\begin{aligned} I &= I_h \cdot 2^{\frac{n}{2}} + I_l \\ J &= J_h \cdot 2^{\frac{n}{2}} + J_l \end{aligned}$$

4. It follows that,

$$\begin{aligned} I \cdot J &= (I_h \cdot 2^{\frac{n}{2}} + I_l) \cdot (J_h \cdot 2^{\frac{n}{2}} + J_l) \\ &= \end{aligned}$$

Integer Multiplication

Main Issues

1. Two n-bit numbers I and J can be added and subtracted in $O(n)$ time.
2. Standard multiplication takes $O(n^2)$ time. (Why?)
3. Can we do better? Observe that,

$$\begin{aligned} I &= I_h \cdot 2^{\frac{n}{2}} + I_l \\ J &= J_h \cdot 2^{\frac{n}{2}} + J_l \end{aligned}$$

4. It follows that,

$$\begin{aligned} I \cdot J &= (I_h \cdot 2^{\frac{n}{2}} + I_l) \cdot (J_h \cdot 2^{\frac{n}{2}} + J_l) \\ &= I_h \cdot J_h \cdot 2^n + I_l \cdot J_h \cdot 2^{\frac{n}{2}} + I_h \cdot J_l \cdot 2^{\frac{n}{2}} + I_l \cdot J_l \end{aligned}$$

Integer Multiplication

Main Issues

1. Two n-bit numbers I and J can be added and subtracted in $O(n)$ time.
2. Standard multiplication takes $O(n^2)$ time. (Why?)
3. Can we do better? Observe that,

$$\begin{aligned} I &= I_h \cdot 2^{\frac{n}{2}} + I_l \\ J &= J_h \cdot 2^{\frac{n}{2}} + J_l \end{aligned}$$

4. It follows that,

$$\begin{aligned} I \cdot J &= (I_h \cdot 2^{\frac{n}{2}} + I_l) \cdot (J_h \cdot 2^{\frac{n}{2}} + J_l) \\ &= I_h \cdot J_h \cdot 2^n + I_l \cdot J_h \cdot 2^{\frac{n}{2}} + I_h \cdot J_l \cdot 2^{\frac{n}{2}} + I_l \cdot J_l \end{aligned}$$

5. So a natural divide and conquer algorithm suggests itself.

Integer Multiplication

Main Issues

1. Two n-bit numbers I and J can be added and subtracted in $O(n)$ time.
2. Standard multiplication takes $O(n^2)$ time. (Why?)
3. Can we do better? Observe that,

$$\begin{aligned} I &= I_h \cdot 2^{\frac{n}{2}} + I_l \\ J &= J_h \cdot 2^{\frac{n}{2}} + J_l \end{aligned}$$

4. It follows that,

$$\begin{aligned} I \cdot J &= (I_h \cdot 2^{\frac{n}{2}} + I_l) \cdot (J_h \cdot 2^{\frac{n}{2}} + J_l) \\ &= I_h \cdot J_h \cdot 2^n + I_l \cdot J_h \cdot 2^{\frac{n}{2}} + I_h \cdot J_l \cdot 2^{\frac{n}{2}} + I_l \cdot J_l \end{aligned}$$

5. So a natural divide and conquer algorithm suggests itself. (Can you design it?)

Integer Multiplication

Integer Multiplication

Analysis

Integer Multiplication

Analysis

Let $T(n)$ denote the running time to multiply two n-bit numbers.

Integer Multiplication

Analysis

Let $T(n)$ denote the running time to multiply two n-bit numbers. It follows that,

Integer Multiplication

Analysis

Let $T(n)$ denote the running time to multiply two n-bit numbers. It follows that,

$$T(n) \quad =$$

Integer Multiplication

Analysis

Let $T(n)$ denote the running time to multiply two n-bit numbers. It follows that,

$$T(n) = \begin{cases} O(1), & \text{if n=1} \\ \end{cases}$$

Integer Multiplication

Analysis

Let $T(n)$ denote the running time to multiply two n-bit numbers. It follows that,

$$T(n) = \begin{cases} O(1), \text{ if n=1} \\ 4 \cdot T(\frac{n}{2}) + b \cdot n, \text{ otherwise} \end{cases}$$

Integer Multiplication

Analysis

Let $T(n)$ denote the running time to multiply two n-bit numbers. It follows that,

$$T(n) = \begin{cases} O(1), \text{ if n=1} \\ 4 \cdot T(\frac{n}{2}) + b \cdot n, \text{ otherwise} \end{cases}$$

It follows that $T(n) \in O(n^2)$!

New Approach

New Approach

Insights

New Approach

Insights

Observe that,

New Approach

Insights

Observe that,

$$(I_h - I_l) \cdot (J_l - J_h) =$$

New Approach

Insights

Observe that,

$$(I_h - I_l) \cdot (J_l - J_h) = I_h \cdot J_l - I_l \cdot J_l - I_h \cdot J_h + I_l \cdot J_h$$

Next note that,

New Approach

Insights

Observe that,

$$(I_h - I_l) \cdot (J_l - J_h) = I_h \cdot J_l - I_l \cdot J_l - I_h \cdot J_h + I_l \cdot J_h$$

Next note that,

$$I \cdot J =$$

New Approach

Insights

Observe that,

$$(I_h - I_l) \cdot (J_l - J_h) = I_h \cdot J_l - I_l \cdot J_l - I_h \cdot J_h + I_l \cdot J_h$$

Next note that,

$$I \cdot J = I_h \cdot J_h \cdot 2^n + [(I_h - I_l) \cdot (J_l - J_h) + I_h \cdot J_h + I_l \cdot J_l] \cdot 2^{\frac{n}{2}} + I_l \cdot J_l.$$

Let $T(n)$ denote the running time to multiply two n-bit numbers.

New Approach

Insights

Observe that,

$$(I_h - I_l) \cdot (J_l - J_h) = I_h \cdot J_l - I_l \cdot J_l - I_h \cdot J_h + I_l \cdot J_h$$

Next note that,

$$I \cdot J = I_h \cdot J_h \cdot 2^n + [(I_h - I_l) \cdot (J_l - J_h) + I_h \cdot J_h + I_l \cdot J_l] \cdot 2^{\frac{n}{2}} + I_l \cdot J_l.$$

Let $T(n)$ denote the running time to multiply two n-bit numbers. It follows that,

New Approach

Insights

Observe that,

$$(I_h - I_l) \cdot (J_l - J_h) = I_h \cdot J_l - I_l \cdot J_l - I_h \cdot J_h + I_l \cdot J_h$$

Next note that,

$$I \cdot J = I_h \cdot J_h \cdot 2^n + [(I_h - I_l) \cdot (J_l - J_h) + I_h \cdot J_h + I_l \cdot J_l] \cdot 2^{\frac{n}{2}} + I_l \cdot J_l.$$

Let $T(n)$ denote the running time to multiply two n-bit numbers. It follows that,

$$T(n) =$$

New Approach

Insights

Observe that,

$$(I_h - I_l) \cdot (J_l - J_h) = I_h \cdot J_l - I_l \cdot J_l - I_h \cdot J_h + I_l \cdot J_h$$

Next note that,

$$I \cdot J = I_h \cdot J_h \cdot 2^n + [(I_h - I_l) \cdot (J_l - J_h) + I_h \cdot J_h + I_l \cdot J_l] \cdot 2^{\frac{n}{2}} + I_l \cdot J_l.$$

Let $T(n)$ denote the running time to multiply two n-bit numbers. It follows that,

$$T(n) = \begin{cases} O(1), & \text{if n=1} \\ & \end{cases}$$

New Approach

Insights

Observe that,

$$(I_h - I_l) \cdot (J_l - J_h) = I_h \cdot J_l - I_l \cdot J_l - I_h \cdot J_h + I_l \cdot J_h$$

Next note that,

$$I \cdot J = I_h \cdot J_h \cdot 2^n + [(I_h - I_l) \cdot (J_l - J_h) + I_h \cdot J_h + I_l \cdot J_l] \cdot 2^{\frac{n}{2}} + I_l \cdot J_l.$$

Let $T(n)$ denote the running time to multiply two n-bit numbers. It follows that,

$$T(n) = \begin{cases} O(1), & \text{if n=1} \\ 3 \cdot T(\frac{n}{2}) + b \cdot n, & \text{otherwise} \end{cases}$$

New Approach

Insights

Observe that,

$$(I_h - I_l) \cdot (J_l - J_h) = I_h \cdot J_l - I_l \cdot J_l - I_h \cdot J_h + I_l \cdot J_h$$

Next note that,

$$I \cdot J = I_h \cdot J_h \cdot 2^n + [(I_h - I_l) \cdot (J_l - J_h) + I_h \cdot J_h + I_l \cdot J_l] \cdot 2^{\frac{n}{2}} + I_l \cdot J_l.$$

Let $T(n)$ denote the running time to multiply two n-bit numbers. It follows that,

$$T(n) = \begin{cases} O(1), & \text{if n=1} \\ 3 \cdot T(\frac{n}{2}) + b \cdot n, & \text{otherwise} \end{cases}$$

It follows that $T(n) \in O(n^{\log_2 3}) = O(n^{1.585})$.

Matrix Multiplication

Matrix Multiplication

Issues

Matrix Multiplication

Issues

Matrix Multiplication

Issues

1. Assume that we are given two $n \times n$ matrices $\mathbf{X}$ and $\mathbf{Y}$.

Matrix Multiplication

Issues

1. Assume that we are given two $n \times n$ matrices $\mathbf{X}$ and $\mathbf{Y}$.
2. The goal is to compute $\mathbf{Z} = \mathbf{X} \cdot \mathbf{Y}$.

Matrix Multiplication

Issues

1. Assume that we are given two $n \times n$ matrices $\mathbf{X}$ and $\mathbf{Y}$.
2. The goal is to compute $\mathbf{Z} = \mathbf{X} \cdot \mathbf{Y}$. Recall that $Z[i,j] = \sum_{k=1}^{n} X[i,k] \cdot Y[k,j]$.

Matrix Multiplication

Issues

1. Assume that we are given two $n \times n$ matrices $\mathbf{X}$ and $\mathbf{Y}$.
2. The goal is to compute $\mathbf{Z} = \mathbf{X} \cdot \mathbf{Y}$. Recall that $Z[i,j] = \sum_{k=1}^{n} X[i,k] \cdot Y[k,j]$.
3. The naive algorithm requires n^3 scalar multiplications and n^3 scalar additions.

Divide and Conquer Approach

Divide and Conquer Approach

Approach

Divide and Conquer Approach

Approach

Divide and Conquer Approach

Approach

1. Break each matrix into four parts as shown below:

Divide and Conquer Approach

Approach

1. Break each matrix into four parts as shown below:

$$\begin{pmatrix} I & J \\ K & L \end{pmatrix} = \begin{pmatrix} A & B \\ C & D \end{pmatrix} \cdot \begin{pmatrix} E & F \\ G & H \end{pmatrix}$$

Divide and Conquer Approach

Approach

1. Break each matrix into four parts as shown below:

$$\begin{pmatrix} I & J \\ K & L \end{pmatrix} = \begin{pmatrix} A & B \\ C & D \end{pmatrix} \cdot \begin{pmatrix} E & F \\ G & H \end{pmatrix}$$

2. The following equations follow immediately:

Divide and Conquer Approach

Approach

1. Break each matrix into four parts as shown below:

$$\begin{pmatrix} I & J \\ K & L \end{pmatrix} = \begin{pmatrix} A & B \\ C & D \end{pmatrix} \cdot \begin{pmatrix} E & F \\ G & H \end{pmatrix}$$

2. The following equations follow immediately:

$$\begin{aligned} I &= A \cdot E + B \cdot G \\ J &= A \cdot F + B \cdot H \\ K &= C \cdot E + D \cdot G \\ L &= C \cdot F + D \cdot H. \end{aligned}$$

Divide and Conquer Approach

Approach

1. Break each matrix into four parts as shown below:

$$\begin{pmatrix} I & J \\ K & L \end{pmatrix} = \begin{pmatrix} A & B \\ C & D \end{pmatrix} \cdot \begin{pmatrix} E & F \\ G & H \end{pmatrix}$$

2. The following equations follow immediately:

$$\begin{aligned} I &= A \cdot E + B \cdot G \\ J &= A \cdot F + B \cdot H \\ K &= C \cdot E + D \cdot G \\ L &= C \cdot F + D \cdot H. \end{aligned}$$

3. The running time is captured by the recurrence:

Divide and Conquer Approach

Approach

1. Break each matrix into four parts as shown below:

$$\begin{pmatrix} I & J \\ K & L \end{pmatrix} = \begin{pmatrix} A & B \\ C & D \end{pmatrix} \cdot \begin{pmatrix} E & F \\ G & H \end{pmatrix}$$

2. The following equations follow immediately:

$$\begin{aligned} I &= A \cdot E + B \cdot G \\ J &= A \cdot F + B \cdot H \\ K &= C \cdot E + D \cdot G \\ L &= C \cdot F + D \cdot H. \end{aligned}$$

3. The running time is captured by the recurrence:

$$T(n) = 8 \cdot T(\frac{n}{2}) + b \cdot n^2$$

Divide and Conquer Approach

Approach

1. Break each matrix into four parts as shown below:

$$\begin{pmatrix} I & J \\ K & L \end{pmatrix} = \begin{pmatrix} A & B \\ C & D \end{pmatrix} \cdot \begin{pmatrix} E & F \\ G & H \end{pmatrix}$$

2. The following equations follow immediately:

$$\begin{aligned} I &= A \cdot E + B \cdot G \\ J &= A \cdot F + B \cdot H \\ K &= C \cdot E + D \cdot G \\ L &= C \cdot F + D \cdot H. \end{aligned}$$

3. The running time is captured by the recurrence:

$$T(n) = 8 \cdot T(\frac{n}{2}) + b \cdot n^2$$

4. It follows that $T(n) \in O(n^3)$.

Strassen's Algorithm

Strassen's Algorithm

Approach

Strassen's Algorithm

Approach

Strassen’s Algorithm

Approach

1. Compute the following products:

Strassen's Algorithm

Approach

1. Compute the following products:

$$
\begin{aligned}
S_1 &= A \cdot (F - H) \\
S_2 &= (A + B) \cdot H \\
S_3 &= (C + D) \cdot E \\
S_4 &= D \cdot (G - E) \\
S_5 &= (A + D) \cdot (E + H) \\
S_6 &= (B - D) \cdot (G + H) \\
S_7 &= (A - C) \cdot (E + F)
\end{aligned}
$$

Strassen's Algorithm

Approach

1. Compute the following products:

$$
\begin{aligned}
S_1 &= A \cdot (F - H) \\
S_2 &= (A + B) \cdot H \\
S_3 &= (C + D) \cdot E \\
S_4 &= D \cdot (G - E) \\
S_5 &= (A + D) \cdot (E + H) \\
S_6 &= (B - D) \cdot (G + H) \\
S_7 &= (A - C) \cdot (E + F)
\end{aligned}
$$

2. Verify the following identities:

$$
\begin{aligned}
I &= S_5 + S_6 + S_4 - S_2 \\
J &= S_1 + S_2 \\
K &= S_3 + S_4 \\
L &= S_1 - S_7 - S_3 + S_5
\end{aligned}
$$

Analysis of Strassen’s algorithm

Analysis of Strassen's algorithm

Analysis

Analysis of Strassen's algorithm

Analysis

Let $T(n)$ denote the running time of Strassen's algorithm.

Analysis of Strassen's algorithm

Analysis

Let $T(n)$ denote the running time of Strassen's algorithm. It follows that,

Analysis of Strassen's algorithm

Analysis

Let $T(n)$ denote the running time of Strassen's algorithm. It follows that,

$$T(n) =$$

Analysis of Strassen's algorithm

Analysis

Let $T(n)$ denote the running time of Strassen's algorithm. It follows that,

$$T(n) = 7 \cdot T(\frac{n}{2}) + b \cdot n^2.$$

Analysis of Strassen's algorithm

Analysis

Let $T(n)$ denote the running time of Strassen's algorithm. It follows that,

$$T(n) = 7 \cdot T(\frac{n}{2}) + b \cdot n^2.$$

It follows that $T(n) \in O(n^{\log_3 7}) = O(n^{2.376})$.