

# Midterm 1 Content

## Insertion Sort w/ Sentinel

### Code

```
A[0] ← -∞
for i ← 2 to n do
    t ← A[i]
    j ← i - 1
    while A[j] > t do
        A[j+1] ← A[j]
        j ← j-1
    A[j+1] ← t
```

### Comparisons

$$B = \sum_{i=2}^n 1$$

$$W = \sum_{i=2}^n i$$

$$A = \sum_{i=2}^n \frac{1}{i} \sum_{j=1}^i j$$

### Exchanges

$$B = 0$$

$$W = \sum_{i=2}^n i$$

$$A = \sum_{i=2}^n \frac{1}{i} \sum_{j=1}^i j$$

## Insertion Sort w/out Sentinel

### Code

```
for i ← 2 to n do
    t ← A[i]
    j ← i - 1
    while j > 0 && A[j] > t do
        A[j+1] ← A[j]
        j ← j-1
    A[j+1] ← t
```

### Comparisons

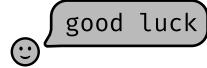
$$B = \sum_{i=2}^n 1$$

$$W = \sum_{i=2}^n (i - 1)$$

$$A = \sum_{i=2}^n \frac{1}{i - 1} \sum_{j=1}^i j$$

### Exchanges

Same as with Sentinel



## Bubble Sort

### Code

```
for i ← n downto 2 do
    for j ← 1 to i - 1 do
        if A[j] > A[j+1] do
            A[j] ↔ A[j+1]
```

### Comparisons

$$\sum_{i=2}^n \sum_{j=1}^{i-1} 1 = \frac{(n-1)(n)}{2}$$

### Exchanges

$$B = 0$$

$$W = \frac{(n-1)(n)}{2}$$

$$A = \sum_{i=2}^n \sum_{j=1}^{i-1} 0.5$$

## Max Continuous Sum

### Code

```
M ← 0; S ← 0
for i ← 1 to n do
    S ← max(S+A[i], 0)
    M ← max(M, S)
```

### Comparisons

$n$

### Exchanges

$0$

## Sort Max-Heap

### Code

```
for i ← ⌊n/2⌋ downto 1 do
    sift(A, i, n)
```

### Comparisons

$$B = \sum_{i=1}^{\lfloor n/2 \rfloor} 2 = n$$

$$W = \sum_{i=1}^{\lfloor n/2 \rfloor} 2 \lg(i) \leq 2 n \lg(n)$$

### Exchanges

$$B = 0$$

$$W = \sum_{i=1}^{\lfloor n/2 \rfloor} \lg(i) \leq n \lg(n)$$

### Exchanges

$B = 0$

## Sift for Max-Heap

### Code

```
func sift(A, i, n) do
    l ← 2i      // Child1
    r ← 2i + 1 // Child2
    M ← i       // Maximum
    if l ≤ n && A[l] > A[i] do
        M ← l
    if r ≤ n && A[r] > A[i] do
        M ← r
    if M ≠ i do
        A[i] ↔ A[M]
        sift(A, M, n)
    return A
```

### Comparisons (i=0)

$B = 2$

$W = 2\lg(n)$

### Exchanges (i=0)

$B = 0$

$W = \lg(n)$

## Max-Heapify Random Array

### Code

```
for i ← ⌊n/2⌋ downto 1 do
    sift(A, i, n)
```

### Comparisons

$$B = \sum_{i=1}^{\lfloor n/2 \rfloor} 2 = n$$

$$W = \sum_{i=1}^{\lfloor n/2 \rfloor} 2 \lg(i) \leq 2 n \lg(n)$$

### Exchanges

$$B = 0$$

$$W = \sum_{i=1}^{\lfloor n/2 \rfloor} \lg(i) \leq n \lg(n)$$

### Funny Log Thing:

$$(a^x)^{\log_a(n)} = n^x$$

## Combination & Permutations

### Combinations with Repetition:

$${}^n C_r = \frac{(n+r-1)!}{r!(n-1)!}$$

### Combinations without Repetition:

$${}^n C_r = \frac{n!}{r!(n-r)!}$$

### Permutations with Repetition:

$${}^n P_r = n^r$$

### Permutations without Repetition:

$${}^n P_r = \frac{n!}{(n-r)!}$$



## Midterm 2 Content

### Quicksort

#### Code

```
func quicksort(s, e, A[]) do
    if s > e do
        exit func
    pi ← (s ... e).rand() or approx_median(A)
    A[pi] ↔ A[s] // Move pivot to start
    q ← partition(s, e, A)
    quicksort(s, q-1, A)
    quicksort(q, e, A)
```

#### Recurrences (Random Pivot)

$$W : T(n) = T(n-1) + T(0) + n - 1$$

$$B : T(n) = 2T(\lfloor n/2 \rfloor) + n - 1$$

$$A : T(n) = \frac{1}{n} \sum_{q=1}^n (T(n-q) + T(q-1)) + n - 1$$

### Selection

#### Code

```
func selection(s, e, k, A[]) do
    if s ≥ e do
        exit func
    pi ← (s ... e).rand() or approx_median(A)
    A[pi] ↔ A[s] // Move pivot to start
    q ← partition(s, e, A)
    if k < q do
        selection(s, q-1, k, A)
    else if k > q do
        selection(q, e, k, A)
    else
        return A[k]
```

#### Recurrences (Random Pivot)

$$W : T(n) = T(n-1) + n - 1$$

$$B : T(n) = T(\lfloor n/2 \rfloor) + n - 1$$

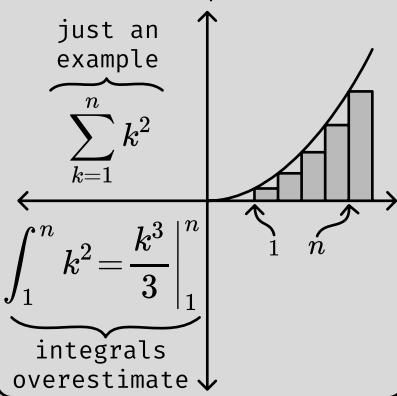
$$A : T(n) = \frac{1}{n} \sum_{q=1}^n (T(n-q)) + n - 1$$

you got this queen

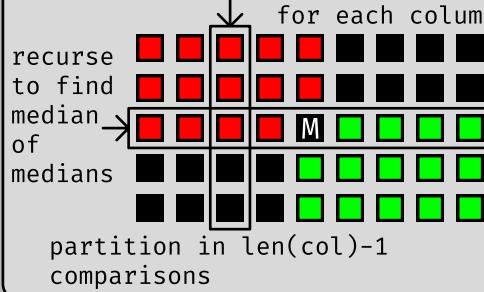


### Bounds and Sums

#### Graph



#### use a known sort (ie bubble) for each column



#### Key:

Key:	Size:
≤ median	$\lfloor \frac{n}{10} \rfloor * 3 \approx \frac{3n}{10}$
≥ median	$\lfloor \frac{n}{10} \rfloor * 3 \approx \frac{3n}{10}$
unkown	$\lfloor \frac{n}{10} \rfloor * 2 * 2 \approx \frac{4n}{10} = \frac{2n}{5}$

### Karatsuba's Fast Multiplication Algorithm

#### Mathematics

ab and cd are numbers of n digits  
 $ab * cd = (10^{n/2}a + b)(10^{n/2}c + d) = 10^n ac + 10^{n/2}(ad + bc) + bd$

$$10^n ac + bd$$

$$\begin{array}{r} \textcolor{purple}{\texttt{TTTT}} \\ + \textcolor{red}{\texttt{0000}} \\ \hline \textcolor{blue}{\texttt{0000}} \end{array}$$

$$+ \begin{array}{r} \textcolor{green}{\texttt{UUUU}} \\ + \textcolor{blue}{\texttt{WWWW}} \\ \hline \textcolor{purple}{\texttt{TTTT}} \end{array}$$

$$+ \begin{array}{r} \textcolor{blue}{\texttt{UUUU}} \\ + \textcolor{yellow}{\texttt{WWWW}} \\ \hline \textcolor{purple}{\texttt{UUUU}} \end{array}$$

(no addition)

$$10^{n/2}(ad + bc)$$

$$\begin{array}{r} \textcolor{cyan}{\texttt{00}} \\ + \textcolor{red}{\texttt{00}} \\ \hline \textcolor{blue}{\texttt{00}} \end{array}$$

$$+ \begin{array}{r} \textcolor{red}{\texttt{XX}} \\ + \textcolor{black}{\texttt{XX}} \\ \hline \textcolor{blue}{\texttt{XX}} \end{array}$$

$$+ \begin{array}{r} \textcolor{magenta}{\texttt{YY}} \\ + \textcolor{black}{\texttt{YY}} \\ \hline \textcolor{blue}{\texttt{YY}} \end{array}$$

$\alpha n$  additions

$$\begin{array}{r} \textcolor{purple}{\texttt{TTTT}} \\ + \textcolor{red}{\texttt{0000}} \\ \hline \textcolor{blue}{\texttt{00}} \end{array}$$

$$+ \begin{array}{r} \textcolor{green}{\texttt{UUUU}} \\ + \textcolor{blue}{\texttt{WWWW}} \\ \hline \textcolor{purple}{\texttt{UUUU}} \end{array}$$

$$+ \begin{array}{r} \textcolor{blue}{\texttt{UUUU}} \\ + \textcolor{yellow}{\texttt{WWWW}} \\ \hline \textcolor{purple}{\texttt{UUUU}} \end{array}$$

$$+ \begin{array}{r} \textcolor{magenta}{\texttt{YY}} \\ + \textcolor{black}{\texttt{YY}} \\ \hline \textcolor{blue}{\texttt{YY}} \end{array}$$

$\alpha n$  additions

Recurrence  
 $T(n) = 4T(n/2) + 2\alpha n$   
 $T(1) = \mu$

Solution  
 $\sum_{i=0}^{\lg(n)-1} (4^i (2\alpha \frac{n}{2^i})) + \mu 4^{\lg(n)}$

#### Even Splitting

$$\sum_{k=1}^n k^2 = \sum_{k=1}^{n/2} k^2 + \sum_{k=n/2+1}^n k^2$$

#### Optimal Splitting

$$\sum_{k=1}^n k^2 = \sum_{k=1}^s k^2 + \sum_{k=s+1}^n k^2$$

Expand and derivate with respect to  $s$ ; set result equal to 0 to find local minima / maxima / optimal bounds.

#### Methodology

In all of these cases, replace the value being summed with the lower or upper bound to obtain approximations.

## Midterm 2 Content - Continued

### Merge

#### Code

```
func merge(A[], B[]) do
    C ← []
    while len(A) > 0 && len(B) > 0 do
        if A.head() ≤ B.head() do
            C.append(A.head())
            A.drophead()
        else do
            C.append(B.head())
            B.drophead()
        while len(A) > 0 do
            C.append(A.head())
            A.drophead()
        while len(B) > 0 do
            C.append(B.head())
            B.drophead()
    return C
```

Comparisons ( $\text{len}(A)=\text{len}(B)=n/2$ )

$$B = \sum_{i=1}^{n/2} 1 = n/2$$

$$W = \sum_{i=1}^{n-1} 1 = n - 1$$

### Mergesort

#### Code

```
func mergesort(s, e, A[]) do
    if s ≥ e do
        exit func
    c = ⌊(s+e)/2⌋
    X ← mergesort(s, c, A[])
    Y ← mergesort(c+1, e, A[])
    return merge(X, Y)
```

#### Recurrences

$$B : T(n) = 2T(n/2) + n/2$$

$$W : T(n) = 2T(n/2) + n - 1$$

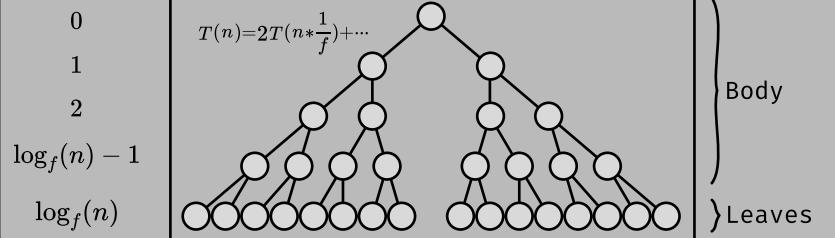
#### Solution (worst case)

$$\sum_{i=0}^{\lg(n)-1} 2^i \left(\frac{n}{2^i} - 1\right) = n \lg(n) - n + 1$$

 you are slay mama boots

### Layer

### Tree Method



### Bucketsort

#### Code

```
func bucketsort(A[]) do
    for i in (1 ... n) do
        solve A[i]'s bucket
        store A[i] in bucket
    concatenate bucket contents
```

Comparisons & Exchanges

0

### Radixsort

#### Code

```
func radixsort(A[]) do
    for each digit
        bucketsort(digit)
```

Comparisons & Exchanges

0

### Countingsort

#### Code (values in $(0 \dots k-1)$ )

```
func countingsort(A[]) do
    C = array of k zeros
    for i in (1 ... n) do
        C[A[i]] += 1
    t ← 0
    for i in (0 ... k-1) do
        C[i] ← C[A[i]] + 1
    for j in (n ... 1) do
        B[C[A[j]]] ← A[j]
        C[A[j]] ← C[A[j]] - 1
    return B
```

Comparisons & Exchanges

0

## Dijkstra

## Goal

Discover the shortest path between an origin point and all other nodes in a weighted graph. AKA The Shortest Path Tree.

## Code

```
func dijkstra(G, s) do
    dist ← [∞, n times]
    pred ← [NULL, n times]
    S ← []
    dist[s] ← 0
    while len(S) ≠ n do
        x ← vertex in G not including S with lowest weight
        for each edge attached to x do
            y ← the connected vertex
            if dist[x] + edge.weight < dist[y] do
                dist[y] ← dist[x] + (Weight of Edge x,y)
                pred[y] ← x
        S.append(x)

    return pred
```

## Adjacency Matrix

$$T(G) = \Theta(V(2V)) = \Theta(V^2)$$

## Adjacency List

$$T(G) = \Theta(V(V+E)) = \Theta(VE)$$

## Prim

## Goal

Discover the Minimum Spanning Tree (MST) of a weighted graph.

## Code

```
func prim(G) do
    let s ← arbitrary starting vertex
    T ← []
    U ← [s]
    while U ≠ V do
        (u, v) ← lowest cost edge where u ∈ U and v ∈ (V - U)
        T ← T ∪ (u, v)
        U ← U ∪ v
```

## Adjacency Matrix

$$T(G) = O(V^2)$$

## Min-Heap for Edges

$$T(G) = O(V \lg(E))$$

## Kruskal

## Goal

Discover the Minimum Spanning Tree (MST) of a weighted graph.

## Code

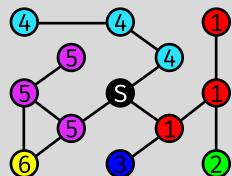
```
func kruskal(G) do
    let EX ← the edges of the graph, sorted
    T ← []
    for each edge (u, v) in EX do
        if u and v are not part of the same tree do
            T.append((u, v))
```

## Runtime

For a graph with E edges and V vertices, Kruskal's algorithm can be shown to run in  $O(E \log E)$  time, or equivalently,  $O(E \log V)$  time, all with simple data structures.

## Depth First Search

## Graph



## Properties

visited = [False, n times]

```
function DFS(G,x):
    visited[x] ← True
    for each vertex y adjacent x do
        if visited[y] = False do
            DFS(G,y)
```

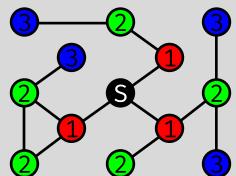
DFS(G,s)

## Run Time

$$O(V) + \Theta(V) + \Theta(E) = O(V + E)$$

## Breadth First Search

## Graph



## Code

```
func BFS(G, s) do
    // Unprocessed Vertices
    queue ← [s]
    // Discovered Vertices
    D ← [False, n times]
    D[s] ← TRUE
    while len(queue) > 0 do
        u = queue.pop()
        for each vertex v adjacent u do
            if D[v] = False do
                queue.push(v)
                D[v] ← True
                // Check something
```

## Run Time

$$O(V) + \Theta(V) + \Theta(E) = O(V + E)$$

# Final Content

**Parallel Analysis**

**Speedup and Efficiency**

$$S_p(N) = \frac{T_1(N)}{T_P(N)}$$

$$E_p(N) = \frac{S_p(N)}{P} = \frac{T_1(N)}{PT_P(N)}$$

An algorithm is only considered "Efficient" iff

$$E_p(N) = \Theta(1)$$

Note:

$$1 \leq S_p(N) \leq P$$

$$\frac{1}{P} \leq E_p(N) \leq 1$$

**Bloom Filters**

**Visualization**

**Analysis (x=existing elements)**

Bit is Zero  
 $(1 - 1/s)^{kx}$

**NP**

**Show NP**

To show that an algorithm is in NP, the simplest methodology is to show that verifying a possible solution, called a 'certificate', can be done in polynomial time.

**Decision Version**

The decision version of an optimization problem should be a task of comparable computational complexity, yet only **outputs** a Yes/No response. Its input is identical to the optimization version, seldom a variable k which represents the axis being optimized.

To show that decision follows from optimization time, use the output of the optimization problem, determine the k associated with it, knowing this is the optimal k. Finally showcase that this output is correct for the given k.

**Optimization Version**

The optimization version of a problem **outputs** the actual solution in question. Its input is identical to decision, seldom the variable k which is absent.

To show that optimization follows from decision, use the output of the decision problem over and over to determine the optimal k. Then, run the decision version with that constant k and a modified input over and over until the true solution can be deduced.

**Zobrist Hashing**

**Methodology**

Assign a unique bit string identifier to each unique board piece. When the state of one board piece changes, XOR it's existing position with the existing hash to remove its existing position from the hash by the magical property of XOR. Then, XOR its new position with the hash to get the new board state. Setup may be expensive but all future board moves will run in constant time ( $2 * \text{XOR}$ ).

**Random Numbers**

**Linear Congruential Generator**

This methodology is limited because a, c, and m must all be chosen very carefully. Even still, the tree of possible outcomes may not be evenly weighted, leading some outputs to be more prevalent than others by necessity.

$$x_{n+1} = ax_n + c \bmod m$$

**Middle Square**

Simply square a number and use its middle digits as the seed for the next number in the sequence.

$$x_{n+1} = \left(\frac{x_n^2}{2^d}\right) \bmod d$$