CMPT 280 ASSIGNMENT 2

Tightest Upper Bound — Is an upper bound on the time/space it will take for a given computation runtime.

1 (a) $f_1(n) = n\log_2 n + n^4 \log_{280} n + 2^n/42 = O(2^n)$

Soln

The computational runtime is $O(2^n)$ because $2^n/42$ is the higher bound in the expression

(b) $f_3(n) = 0.4n^4 + n^2 \log n^2 + \log_2(2^n) = O(n^4)$

Soln

The computational runtime is $O(n^4)$ because $0.4n$ is the tightest upper bound in the expression

(c) $f_2(n) = 4n^{0.7} + 29n\log_2 n + 280 = O(n\log n)$

Soln

The computational runtime is $O(n\log n)$ because $29n\log_2 n$ is the higher/upper bound in the expression

2. $T_A(n) = \frac{1}{280} n^2 + 42\log n + 12n^3 + 280\sqrt{n}$

1. Algorithm A is $O(\log n)$ → FALSE
2. Algorithm A is $O(n^2)$ → FALSE
3. Algorithm A is $O(n^3)$ → TRUE
4. Algorithm A is $O(2^n)$ → TRUE

b. Time complexity of algorithm A is $O(n^3)$

3. i. $O(n^2) + O(\log n) + O(n \log n)$

$\Rightarrow O(\max(n^2, \log n)) + O(n \log n)$

$\Rightarrow O(n^2) + O(n \log n)$

$\Rightarrow O(\max(n^2, n \log n))$

$\Rightarrow O(n^2)$ //

[linear expression is bigger than logarithmic expression]

ii. $O(2^n) \cdot O(n^2)$

$\Rightarrow O(2^n \cdot n^2)$

$\Rightarrow O(2^n n^2)$ //

iii. $420(n \log n) + 180(n^3)$

$\Rightarrow O(420 n \log n) + O(180(n^3))$

$\Rightarrow O(\max(n \log n, n^3))$

$\Rightarrow O(n^3)$ //

iv. $O(n^2 \log_2 n^2) + O(m)$ [m is independent of n]

$\Rightarrow O(n^2 \log_2 n^2 + m)$

$\Rightarrow O(n^2 \log_2 n^2 + m)$ //

(4) Consider the following java code fragment

```
for(i=1; i<=n; i++){
    for(j=1; j<=n; j++){
    System.out.println(i + ", " + j);
}
```

3

Outer Loop

Number of Loop Body statement = 1 times

Execution = $n+1$

Total cost = $2n^2 - 2n$

$$c \cdot \sum_{i=1}^{n+1} = c \cdot \frac{(n+1)n}{2} = \frac{c}{2}(n^2+n) \in \Theta(n^2)$$

Inner Loop

Number of Loop Body statement = 2 times

Execution = $n+1$

Total cost = $2n+1$

(6) $T(n) = 2n^2 - 2n$

Big O $\Longrightarrow O(2n^2) - O(2n)$

$\Longrightarrow O(\max(n^2, n))$

$\Longrightarrow O(n^2)$ //

(5)

```
n = a.length
for i = 0 to n-1
    for j = i+1 to n-1
        print a[i] + " duels " + a[j] + ", yarr!"
```

Soln

Outer Loop

Number of Loop Body statement = 1 statement each time

Execution = n

Inner Loop
Number of Loop Body statement = 2 times
Execution = $n-1$

Total cost =
Total statements = 3 times

$1 + 3 * (n-1)(n)/2$ [3 times because
1 [Print statement]
2 [for loop]

$1 + \frac{3n}{2}(n-1)$

$\Longrightarrow 1 + \frac{3}{2}n(n-1)$ $3/2 \Longrightarrow 1.5$

$\Longrightarrow 1 + 1.5n(n-1)$
$1 + 1.5n^2(n^2-n) \rightarrow O(n)$

Big O $\Longrightarrow O(1) + O(n^2) - O(n)$
$O(\max(n^2, n))$
$\Longrightarrow O(n^2)$
∴ The Upper Bound is $O(n^2)$ //

6. Using Active Operation approach
$1 + 3/2\, n(n-1)$ $\Rightarrow 3/2 = 1.5$
$1 + 1.5n(n-1)$
$1 + 1.5(n^2-n)$
$1 + 1.5n^2 - 1.5n$
Getting the upper bound using polynomial hierarchy [K]
$1 + kn^2 - kn$
$\Longrightarrow O(1) + kO(n^2) - kO(n)$
$\Longrightarrow O(\max(n^2, n))$
$\Longrightarrow O(n^2)$ Upper Bound $\Longrightarrow O(n^2)$ //

(7) Using the active operation to timing analysis; determine the time complexity of this pseudocode in worst case.

Soln

Worst case = nth array in m items

The time complexity of the worst case will be O(log m) using the binary search

Binary Search

Binary worse complexity = O(log N)

Average performance = O(log N)

Best case = O(1)

O(log m is the algorithm of the array) because the array has m items

For Example:- Consider this binary search in 2D array

0	1	2	3	4	5	6	7	8	9
1	2	3	4	5	6	7	8	9	10
1	2	3	4	5	6	7	8	9	10

Is the Lowest

Is the highest Bound

∴ The worst case will be the nth term and if the nth term in the array of the binary search will be sorted

$\Longrightarrow O(n) * O(\log m)$

$\Longrightarrow$ Big O $(O\ n * \log m)$

$\Longrightarrow \Theta(n \log m)$ //

Worst complexity = $O(n \log m)$ //

ADT Specifications

(8) Name: Queue<G>

Sets:
S: is set of all queues containing elements from G
G: is set of priority items that can be in the queue S
B: {True, False}
I: is set of non-negative integers

Signatures:
newPriorityQueue<G>(x): $I \nrightarrow S$
S.insert(y): $G \rightarrow S$
S.isEmpty: $\rightarrow B$
S.isFull: $\rightarrow B$
S.maxItem: $\nrightarrow G$
S.minItem: $\nrightarrow G$
S.deleteMax: $\nrightarrow S$
S.deleteMin: $\nrightarrow S$
S.deleteAllMax: $\nrightarrow S$
S.frequency: $\nrightarrow I$

Pre conditions:- $\forall s \in S, y \in G, x \in I$

newPriorityQueue<G>(x): $x > 0$
S.insert(y): Not full (S is not full)
S.isEmpty: None
S.isFull: None
S.maxItem: S is not empty
S.minItem: S is not empty
S.deleteMax: S is not empty
S.deleteAllMax: S is not empty
S.deleteMin: S is not empty
S.frequency: Is None

Semantics:- $\forall s \in S, y \in G, x \in I$

newPriorityQueue<G>(x): It creates a new queue that holds the set of elements from G with the capacity x

S.insert(y): Adds y to the element of S with a certain priority

S.isEmpty: return True if s is empty, false otherwise

S.isFull: return True if s is full, false otherwise

S.maxItem: return the item element of s with the highest priority

S.minItem: returns the item of s with the lowest priority

S.deleteMax: It deletes the item s for the highest priority

S.deleteAllMax: It deletes/removes s from all the items that are tied for the highest priority

S.deleteMin:- It removes s from the item with lowest priority

S.frequency:- returns the number of items obtained from the certain item in s with any priority.