Elizabeth Oyebade

MET CS 526-O2

03/29/2022

Problem 1.1

```
x = 0;    // O(1)
y = 0;    // O(1)
for (i=1; i<n; i++)
    …     // O(n-1)
return (x - y); // O(1)
```

In this loop, what's shown is the value i will go from 1 to n. Where n is the size of the array, the value n will increase linearly. Therefore, analyzing the running time complexity will be O(n).

Problem 1.2

```
x = 0;
i = 0;
while (i < n) {
y = 0;
j = 0;
while (j < n) {
k = 0
while (k <= j) {
y = y + a[k];
k = k + 1;
}
j = j + 1;
}
 if (b[i] == y) {
x++;
} i = i + 1;
}
return x;
```

There are three while-loops, and they can be described as a nested loop.
The outer first while loop, the value of i will run from 0 to n-1 so, it will run for n times.
The inner second while loop will also run for n times but for every iteration of the first loop. So, for each value of i, the value of j will move from 0 to n-1.
The inner third while loop will for j times for each iteration of the second loop. For each value of j, the value k moves from 0 to j depending on the value of j.
The third loop will run for 1 time when j=0, 2 times when j=1, 3 times when j=2, …, n times when j=n.
It will run for: (1+2+3+4+5+…+n) times = n*(n+1)/2 times.
Being that the first loop will run for n iteration, the second and third loop will combinedly run for n*(n+1)/2. Therefore, the analyzed running time complexity will be n*n*(n+1)/2 which is $O(n^3)$.

Problem 1.3

```
if (i == 0) { //O(1)
 p[0] = a[0];
p[1] = a[0];
}
else {
method3(a, i-1, p); //O(n-1)
if (a[i] < p[0]]) { //O(1)
p[0] = a[i];
}
if (a[i] > p[1]]) { //O(1)
p[1] = a[i];
}
}
```

The starting value of i is n-1. With the new i value as i-1, the recursive calls are being made. This function is being called recursively n times before reaching the base case so the analyzing running time complexity will be O(n).

Problem 1.4

```
if (x >= y) { //O(1)
return a[x];
}
else {
z = (x + y) / 2; //O(1)
u = method4(a, x, z); //O(n)
v = method4(a, z+1, y); //O(n)
if (u < v) return u; //O(1)
else return v;
}
}
```

The analyzed running time complexity will be O(n) because each node will be split into 2 nodes also known as the child nodes. So, it will be equal to the number of the internal nodes (2n-1) as a function will be called for 2n-1 times. Therefore, using the Master theorem, $T(n)= a.T(n/b)+O(n^d)$ where a=2, b=2, d=0 as $a>b^d$ there by making the running time complexity $O(n^{(\log_2 2)})$ which is O(n)

Problem 2.1

Operation	Return Value	Stack Contents
push(10)	-	(10)
pop( )	10	()
push(12)	-	(12)
push(20)	-	(12,20)
size( )	2	(12,20)
push(7)	-	(12,20,7)
pop( )	7	(12,20)
top( )	20	(12,20)
pop( )	20	(12)
pop( )	12	()
push(35)	-	(35)
isEmpty( )	false	(35)

Problem 2.2

Operation	Return Value	Queue Contents (first ← Q ← last)
enqueue(7)	-	(7)
dequeue( )	7	()
enqueue(15)	-	(15)
enqueue(3)	-	(15,3)
first( )	15	(15,3)
dequeue( )	3	(15)
dequeue( )	15	()
first( )	null	()
enqueue(11)	-	(11)
dequeue( )	11	()
isEmpty( )	true	()
enqueue(5)	-	(5)