Data Structures and Algorithms

DIT 181

Assignment 1

05/02/2018

Group 9

Team Members:

1. Anders Karlsson
2. George Sarkisian
3. Majed Dalain

Number of pages in the assignment:

pages

6

+

two java files

Dynamic arrays

Question 1:
the comlexuty of reverse is n/2 and in big O notation is O(n/2)
‘check the class Array’

Question 2:
the complexity of this is n and in big o notation is O(n),
‘check the class Array’

Question 3:
the complexity of the method is n-1-i and big O notation is O(n).
And in removeFast you can see that in this way we swap the last element the element that we want to remove and complexity of this is O(1).
‘check the class Array’

Question 4:
the complexity of the method is o(n) and it can’t be more officiant since the array is not sorted and we have to go throw all array to search.
‘check the class Array’

Question 5:
instead of checking the half of the array is empty we can check if the quarter of the array is empty and after that delete the half of it and in that way we will save space and save the time complexity

Question 6:
the complexity of this method is O(n^2) and it can’t do better since the array is not sorted and the we have a loop that goes throw the array and another loop that goes that goes right and left of that selecting element in the array to find the palindrome.
‘check the class Array’

Complexity

Question 7

7.1) $\sum_{i=1}^{n} i = 1+2+3+...+n$ which is equal to $(n(n+1)) / 2$ according to the sum identity of an arithmetic series. This means that the function has the big-O complexity O(n^2).

7.2) $\sum_{i=1}^{n} i^2 = 1+4+9+...+n^2$ which is equal to $(n(n+1)(2n+1)) / 6$ according to the square pyramidal numbers. This means that the function has the big-O complexity O(n^3)

7.3) $\sum_{i=1}^{n} 2^i = 2+4+8+...+2^n$ which is equal to $(2(1-2^n)) / (1-2)$ as it is a finite geometric series. This means that the function has the big-O complexity O(2^1+n)

7.4) $\sum_{i=1}^{lg(n)} 2^i = 2+4+8+...+2^{lg(n)} = 2+4+8+...+n$. This means that the function has the big-O complexity O(2n) as the function keeps multiplying by 2.

7.5) $\sum_{i=1}^{n} 1/i = 1+1/2+1/3+...+1/n = Hn$ which represents the n:th harmonic number. This means that the function has the big-O complexity O(log(n))

Question 8

Show that $4^n \notin O(2^n)$

We assume that $4^n \in (2^n)$ which means we have two constants, C and N.
For n >= N we get $4^n <= C * 2^n$ which means that $2^n(C-2^n) >= 0$, since $C-2^n$ goes towards $-\infty$ we have proved that the original statement is true via proof by contradiction.

Question 9

$a_0+a_1x+a_0x^2+...+a_{n-1}x^{(n-1)}+a_nx^n$

One way to speed up the calculation of x^n is to use a divide & conquer method e.g. x^80 = x^40*x^40, x^40=x^20*x^20 which would result in less operations and have the complexity of O(log(n)).

Given the polynomial in our case we can apply Horner's method which tells us that we can rewrite the polynomial as $f(x)=a_0+x(a_1+x(a_2+...+x(a_{n-1}+a_nx)))$ which would have the complexity of O(n).

Recurrences, divide and conquer

Question 10

Consider the following code:

```
import java.math.BigInteger;
public static BigInteger fib(int n) {
if (n <= 1) return BigInteger.ONE;
return fib(n-1).add(fib(n-2));
}
```

What is the complexity of this function assuming that each addition is $O(1)$?

Hint: What is the relationship between the run time and the result returned by the function?

Propose a faster implementation of the same function, and give its complexity.

Since the complexity for addition is of type O(1), and as we know that the complexity of the traditional recursive call for the Fibonacci is of type exponential one, so the complexity of this function is O(2^n).

There are many faster implementation for the Fibonacci function, we could apply the tail recursive algorithm where we calculate the results before we do the recursive call, and the complexity of this algorithm is O(n) from the operations perspective, the following is the code for this algorithm:

```
int fib(int term, int val = 1, int prev = 0) {

 if(term == 0) return prev;

if(term == 1) return val;

return fib(term - 1, val+prev, val);

}
```

Question 11

* T(n)=T(n-1)+0(1/n)

T(n-1) = T(n-1-1) +(1/n-1)

T(n-2) = T(n-1-1-1)+(1/n-1-1)

T(n-2) = T(n-1-1-1-1)+(1/n-1-1-1)

for K = 1 ⇒ [T(n-2)+1/n-1] +1/n

for K = 2 ⇒ [T(n-3)+1/n-2] + 1/n-1 + 1/n

⇒ T(n-k) + 1/n-k ……..∑1/i so the complexity here is Θ(log n)

*$T(n) = 3T(n/2) + \Theta(n^2)$

By applying the master method where a T(n/b) + n^c we got a = 3, b=2 and c=2

b^c = 4 > a → T(n) = Θ(n^c) → Θ(n^2)

*$T(n) = 4T(n/2) + \Theta(n^2)$

By applying the master method where a T(n/b) + n^c we got a = 4, b=2 and c=2

b^c = 4 = a → T(n) = Θ(n^c log b (n)) → Θ(n^2 log2 (n))

*$T(n) = 5T(n/2) + \Theta(n^2)$

By applying the master method where a T(n/b) + n^c we got a = 5, b=2 and c=2

b^c = 4 < a → T(n) = Θ(n^log b (a)) → Θ(n^log2 (5)) = (n^2.3)

Question 12

‘Check the class Array’

Question 13

‘Check the class Array’

Question 14

‘Check the class Array’

Sorting

Question 15

‘Check the class lab1Sorting’