Problem Set 2

CSCI 3104 Spring 2014
Cristobal Salazar
07/22
Partner: Alex Tsankov

Problem 1

a) The best case is $\Theta(1)$. If the target is halfway through the array a, and m will always be initialized to the point halfway through a, then it will be constant time to complete the function.

b) The worst case is $\Theta(log(n))$. The range of possible numbers is cut in half each loop iteration.

Problem 2

a) To tackle this problem we need to break the list of functions into numbers so we can run our sorting algorithm on it. Deferring to our list in Part B, we can attribute positional values to each of the functions based on running time. For example, position 1, or A_1 would be 1, because that has the fastest running time. $A_2 = n^{(\frac{1}{lg(n)})}$ ... $A_{12} = n!$. Assuming our positioning is right, i.e. Part 2, we can simply run quicksort on A_1 to A_{12} and log the pivot each time. See the attached document for our source code for the quicksort algorithm. Our pivots were as follows: $p_1 = 9$, $p_2 = 4$, $p_3 = 1$, $p_4 = 2$, $p_5 = 8$, $p_6 = 5$, $p_7 = 7$, $p_8 = 9$, $p_9 = 11$. See Part B for our final global array ordering.

b) The sorted list:

1	$n^{1/lg(n)}$	$\sqrt{2}^{lg(n)}$	n	$2^{lg(n)}$	$n * lg(n)$	n^2	$lg(n!)$	$lg(n)!$	$(3/2)^n$	e^n	$n!$

To show that the above statement is true, with $g_x = \Omega(g_{x+1})$, we will compare each of the values to its proceeding value.

1) We can show that 1 is less than $n^{(\frac{1}{lg(n)})}$ using L'hospitals rule. If we apply L'Hospitals rule as n approaches infinity we can see that $n^{(\frac{1}{lg(n)})}$ is bigger.

2) We can show that $\sqrt{(2)}^{lg(n)}$ is greater than $n^{(\frac{1}{lg(n)})}$ using L'hospitals rule. If we apply L'Hospitals rule as n approaches infinity we can see that $\sqrt{(2)}^{lg(n)}$ is bigger.

3) We can show that $\sqrt{(2)}^{lg(n)}$ is less than $2^{lg(n)}$ using L'hospitals rule. If we apply L'Hospitals rule as n approaches infinity we can see that $2^{lg(n)}$ is the proper O because of the slightly larger constant.

4) We can show that n is greater than $2^{lg(n)}$ using L'hospitals rule as $n \to \infty$ approaches ∞.

5) We can show that $2^{lg(n)}$ is less than $n * lg(n)$ using L'hospitals rule as $n \to \infty$ approaches ∞. This can be shown intutiviely by the fact that $n * lg(n)$ is guarenteed to be bigger than $2^{lg(n)}$ for any $n > 2$

6) $n * lg(n)$ is less than n^2, because intuitively, $lg(n) < n$, if we just take out an n.

7-10) We can can compare these by taking the logarithm of both sides, like Amir showed in lecture

11) We can show that e^n is less than $n!$ using L'hospitals rule. If we apporximate $n!$, to equal n^n, we apply L'Hospitals rule as n approaches infinity of e^n/n^n, we can see that it goes to 0 meaning $n!$ is larger.

Problem 3

a) The recurrence relation is $a_n = 2 * a_{n-1} - 1,$, where $a_0 = 3, a_1 = 5$.

b) To find the characteristic polynomial of $a_n = 2 * a_{n-1} - 1$, we follow the steps from the link given in the problem:

$$t^n = 2 * t^{n-1}$$
$$0 = -t^n/(t^{n-1}) + 2 * t^{n-1}/(t^{n-1})$$
$$0 = -t^{n-n+1} + 2 * t^{n-1-n+1}$$
$$0 = t - 2, \text{ so the root is } t = 2$$

Now to find the characteristic polynomial we plug the root into the form $a_n = A * a^n + C$, given $a_0 = 3$ and $a_1 = 5$

$$a_0 = A * 2^0 + C = 3$$
$$a_1 = A * 2^1 + C = 5$$

We solve the system to get $A = 2$ and $C = 1$, therefore $a_n = 2*2^n+1$. So, $\boldsymbol{a_n = 2^{n+1} + 1}$.

c) Proof that $a_n = 2*a_{n-1}-1$ is equivalent to $a_n = 2^{n+1}+1$, with base cases $a_0 = 3, a_1 = 5$:

$$a_0 = 2^{0+1} + 1 = 2 + 1 = 3$$
$$a_1 = 2^{1+1} + 1 = 4 + 1 = 5$$

The base cases work, therefore we assume $a_0, a_1, ..., a_{n-1}$ are correct.

$$a_n = 2 * (2^{(n-1)+1} + 1) - 1 = 2 * (2^n + 1) - 1$$
$$a_n = 2 * 2^n + 2 - 1 = 2^{n+1} + 1$$

Therefore, $a_n = 2 * a_{n-1} - 1$ is equivalent to $a_n = 2^{n+1} + 1$.

d)The running time of $f(n)$ is given by the recurrence relation $a_n = (\sqrt{a_{n-1}} + 1)^2$, with base cases $a_0 = 1, a_1 = 1$.

e) Each time the function is run, it calls 2 recurrsions, and the value that is put in is incrementing down by a single operation(i.e. no division). This essentially means that if we put in n, we will be doing n^2 operations, that is the tree is squaring each time we call the function. Therefore the upper-bound $O(2^n)$ is trivially large compared to $O(n^2)$.

f)Using the same method as above, we get that $A = 1, B = 1$, and $C = 1/4$, so the recurrence relation can be written as $a_n = n^2+n+1/4$, so a tight upper-bound will be $O(n^2)$

Problem 4

c) To find the big-O of $T(n) = T(n-1) + n$, we unroll the relation. We know $T(n-1) = T(n-2) + n - 1$, and $T(n-2) = T(n-3) + n - 2$. We then plug that in, to get $T(n) = T(n-3) + n - 2 + n - 1 + n = T(n-3) + 3n - 3$. We can see by the pattern that this takes the form $T(n) = T(n-k) + kn - k(k-1)/2$. We also can see that the base case is $n - k = 1$, and so $k = n - 1$, therefore we can substitute in for k to get $T(n) = T(1) + (n-1)n - (n-1)(n-2)/2$, which means the algorithm has $O(n^2)$.

b) To use the Master method for this problem we will go with scenario 1. This entails finding $\mathrm{O}(\mathrm{n}^{\log_b(a)}) - \epsilon$ with $T(n) = \Theta(n^{log_b a})$. $n^3 = \Omega(n^{log_2 2+\epsilon})$ which leads to our final answer: $\Theta(n^3)$

Problem 5

Professor Vector is wrong because her theory breaks down with a tree that is a single node. If node A(the only node), contains value 5, and we are searching for 5, then the sets a, b, and c will be as follows:
$A = \{\}$, nothing to the right of node A
$B = \{A\}$, the path to node A
$C = \{\}$, nothing to the right of node A
This being the case, for $a \in A, b \in B, c \in C$, the claim $a \leq b \leq c$ is false because there are no elements in set A or B to compare.

Sources

- http://stackoverflow.com/questions/13674719/easy-solve-tn-tn-1n-by-iteration-method
    - How to unroll
- http://en.wikipedia.org/wiki/Master_theorem
    - Solving recurrence relations using Master Theorem

Source Code for P2

```
#include <iostream>
using namespace std;

void quickSort(int arr[], int left, int right)
 {
  int i = left, j = right;
  int tmp;
  int pivot = arr[(left + right) / 2];

  while (i <= j) {
        while (arr[i] < pivot)
              i++;
        while (arr[j] > pivot)
              j--;
        if (i <= j) {
              tmp = arr[i];
              arr[i] = arr[j];
              arr[j] = tmp;
              i++;
              j--;
    }
}
cout << "$p_1 = $ " << pivot << "," << endl;
if (left < j)
    quickSort(arr, left, j);
if (i < right)
        quickSort(arr, i, right);
}

int main(){
    cout << "The old array is: ";
    int arr[12]= {5,7,3,4,12,9,10,2,6,8,11,1};
        for(int i=0;i<12;i++)
    cout<<arr[i]<<" " ;
    cout << endl;
    quickSort(arr,0,11);
    cout<< "The new, sorted array is: " << endl;
    for(int i=0;i<12;i++)
    cout<<arr[i]<<" " ;
    cout << endl;

return 0;
}
```