CMSC401 – Fall 2017
HOMEWORK 1
Due 11 am, Oct 3 (Tue class time)
No late submissions

1. Write the recurrence equation for the running time T(n) of the `RecursiveMin` function below. Assume n=r-p+1 is the size of the array A at input.

```
RecursiveMin(A,p,r)
1   if (p==r)
2     return A[p]
3   elseif (p<r)
4     q=floor((p+r)/2)
5     m1=RecursiveMin(A,p,q)
6     m2=RecursiveMin(A,q+1,r)
7     if (m1<m2)
8       return m1;
9     else
10      return m2;
```

Answer:
$T(n) = 2T(n/2) + c$ or
$T(n) = 2T(n/2) + O(1)$

2. What does the following function return (express your answer as a function of n)? Give the worst-case running time using Big-O notation.

```
function something(n)
1   r=0
2   for i=1 to n do
3      for j=1 to i do
4         for k=j to i+j do
5           r=r+1
6   return(r)
```

Answer:
It returns the sum of both the squares of numbers from 1 to n and the numbers themselves
$1^2+2^2+3^2+\ldots n^2 + 1+2+3\ldots n$, which is equal to $n(n+1)(n+2)/3 = O(n^3)$ for $c=1$, $n_0>=2$

3. Show which case is true f(n) = O(g(n)), f(n) = Ω(g(n)), or f(n) = Θ(g(n)) for each of the following function pairs. Justify your answer.

	Answer:
(a) $f(n) = \log n^2$; $g(n) = \log n + 5$	$f(n) = \Theta(g(n))$
(b) $f(n) = n \log n + n$; $g(n) = \log n$	$f(n) = \Omega(g(n))$
(c) $f(n) = 10$; $g(n) = \log 10$	$f(n) = \Theta(g(n))$

(d) $f(n) = \sqrt{n}$; $g(n) = n + \log n$ $f(n) = O(g(n))$

For justification, they need to show c and n0 that will make the definitions true

4. Show that that $n^3-3n^2-n+1 = \Theta(n^3)$. You need to show that there exists c_1 and c_2 defined in Θ definition.

Answer:
We need to show that $c_1n^3 <= n^3-3n^2-n+1 <= c_2n^3$
For upper bound:
$n^3-3n^2-n+1 <= n^3+1 <= c_2n^3$ => $(c2-1)n^3>=1$ is true when **$c_2=2$ $n_0>=2$**
For lower bound:
$c_1n^3 <= n^3-3n^2-n+1$ Select $c_1 = 0.5$
Then, when n>=3 we can write n^3-3n^2-n+1**$-n^2+7$** $<= n^3-3n^2-n+1$
If we can show $c_1n^3 <= n^3-4n^2-n+8$ is true when $c_1=0.5$, lower bound is satisfied.
We need to show when $0.5n^3-4n^2-n+8 >=0$ is true.
- ⇨ $n^2(0.5n-4)-2(0.5n-4) = (n-\sqrt{2})(n+\sqrt{2})(0.5n-4) >=0$
- ⇨ This will be true when **$n_0>=9$ (when $c_1=0.5$)**

Above is trying to add/subtract numbers to make the function decomposable easily. **They can also draw graph of the original function and show c_1 and c_2 directly with n_0.**

5. In INSERTIONSORT, there are two parts maintained in the array: sorted and unsorted. Every time a new element is taken from unsorted side, its correct location in the sorted subarray is found linearly by comparing the elements in the sorted subarray one by one. Can we use binary search to find the correct location of the current element in the sorted subarray to improve the complexity to O(nlogn)?

Answer:
We can reduce the time of finding the correct location for the current element to log(n) but we still need to shift the numbers to the right linearly. Thus, the running time still stays $O(n^2)$.

6. Use the substitution method to show that for the recurrence equation:
$T(n)=T(3n/4) + n/2$
the solution is $T(n)=O(n)$

Answer:
To show $T(n)=O(n)$, we need to show $T(n)<=cn$
Assume this is true for m<n. Select m=3n/4, then
$T(3n/4)<=3cn/4$
$T(3n/4) + n/2 <=3cn/4 +n/2$ (Add n/2 to both sides)
$T(n) <= 3cn/4 +n/2$ (Replace left term with T(n))
$T(n) <= n(3c/4+1/2)$, to show $T(n)<=cn$, we need to show $3c/4+1/2 <= c$, which will be true when c>=2.

7. Use the recursion tree method and find out the solution (T(n) = O(?)) for the following recurrence equation:

T(n)=T(n-1) + 6n

Answer:

They need to draw the tree and show the sum $6(n+(n-1)+\ldots.2+1)=6n(n+1)/2 = O(n^2)$

8. Use Master Method to find solution (T(n) = O(?)) for the following recurrence equations, if possible (You will be given master theorem in the exam)

a) $T(n)=4T(n/2)+n^2$

b) $T(n) = 4T(n/2)+n$

c) $T(n)=4T(n/5)+n \log n$ (with $5^{0.86} = 4$)

Answer:

Case 2 = $O(n^2 \log n)$

Case 1 = $O(n^2)$, there is E>0

Case 3 because $n \log n = \Omega(n^{0.86+E})$, when E=0.1 and regularity condition holds with c=4/5

Give them partial if they do not show regularity condition is satisfied.

9. Function CountEven(A,n) finds the number of occurrences of even integers in n-element array A[1..n]:

```
CountEven(A,n)
1     cnt=0
2     for (i=1;i<=n;i++)
3       if (A[i]%2==0)
4         cnt++
5     return cnt
```

Prove correctness of CountEven using the loop invariant technique. Go through Initialization, Maintenance and Termination parts of the proof. The Loop Invariant is:
"at the start of each iteration of the for loop, variable cnt contains the number of even elements in A[1...i-1]"

Answer: We will assume that it is true at i-th iteration and show it will stay true at (i+1)-th iteration. So, at i-th iteration, we can assume that variable cnt contains the number of even elements in A[1…i-1]. In the (i+1)-th iteration, if A[i] is even, cnt will be increased by 1 and the cnt will still show the number of even elements in A[1…i]. If A[i] is not even, cnt will stay the same and show the number of even elements in A[1…i].

10. Using the numbers from 1 to 9, give a permutation of them (an order of these numbers as an input) that will create the worst case running time for QuickSort? Assume that pivot is selected as the number in the middle (when input size is even, the number at n/2 order is selected. Ex: n=9, pivot=A[5], when n=8 pivot=A[4]) and replaced with last element A[r] first. The rest of the partitioning algorithm is same as it is shown in the slides. Show the running of the QuickSort on that input.

Answer:

9 4 8 6 1 3 5 7 2 (there could be multiple answers)

Steps:

9 4 8 6 1* 3 5 7 2 -> move pivot to end first 9 4 8 6 2 3 5 7 1

1 | 4 8 6 2* 3 5 7 9 -> move pivot to end first 1 | 4 8 6 9 3 5 7 2

1 2 | 8 6 9 3* 5 7 4 -> move pivot to end first 1 2 | 8 6 9 4 5 7 3

1 2 3 | 6 9 4* 5 7 8 -> move pivot to end first 1 2 3 | 6 9 8 5 7 4
1 2 3 4 | 9 8 5* 7 6 -> move pivot to end first 1 2 3 4 | 9 8 6 7 5
1 2 3 4 5 | 8 6* 7 9 -> move pivot to end first 1 2 3 4 5 | 8 9 7 6
1 2 3 4 5 6 | 9 7* 8 -> move pivot to end first 1 2 3 4 5 6 | 9 8 7
1 2 3 4 5 6 7 | 8* 9 -> move pivot to end first 1 2 3 4 5 6 7 | 9 8
1 2 3 4 5 6 7 8 | 9

Above is not the only answer, there could be other possible permutations giving worst case running time. The important part is that the Quicksort should iterate 8 times and each iteration should pick a pivot that is either the min or max of the remaining elements. This will yield a partitioning with all elements in either Part I or Part III (check class notes for part definitions) and reduce the problem to a sub-problem with size 1 less than the previous problem.