Design and Analysis of Algorithms

CS 302 Fall 2020 (Sections 7E, 7F)

# Homework # 1

Total Marks = 83

**Q1)** Perform step-count analysis on the following code fragments. Indicate the time taken by each line of code over the life of the program, then add all individual times to get T(n). Where applicable, work in the worst case scenario. Then find an appropriate O(f(n)) for each T(n). In order to do this, you must show must that there exists a positive constant c>0, such that: T(n) <= c f(n). [5*5 = 25 Marks]

(a)
```
int s, i,n;
cin>>n;
s = 0;
for (i=n;i>=1;i--)
     s++;
```

(b)
```
int sum,i,j,n;
sum = 0;
cin>>n;
for (i=1;i<n;i=i*2)
    for (j=1;j<n;j=j*2)
  sum ++
```

**(c )**

```
int sum, i, j, k,n;
sum = 0;
k=0;
cin>>n;
for (i=0;i<n;++i)
{       k=0;
        sum++;
        for (j=n;j>0;j=j-3)
        {       sum++;
                k=k*sum;
                cout << k;
        }
        cout << k;

}
```

**(d)**

```
int sum,i,j,k,n;
sum = 0;
cin>>n;

for (i=1;i<n;i=i*2)
{       for (j=0;j<n;++j)
            for   (k=1;k<=n;k=k*2)

                sum++;

}
```

**(e )**

```
int i,j,sum,n;
cin>>n;
for (i=1;i<=n;i=i*2)
{       cout << i;
        sum=0;
        for (j=1;j<=i;j=j*2)
        {
                Sum++;
                cout << i;

        }
        cout << Sum;
}
```

**Q2)** Suppose we are comparing implementations of insertion sort and merge sort on the same machine. For inputs of size $n$, insertion sort runs in $4n^2$ steps, while merge sort runs in $32\ n \lg n$ steps. For which values of $n$ does insertion sort beat merge sort? [5 Marks]

**Q3)** What is the smallest value of $n$ such that an algorithm whose running time is $100n^2$ runs faster than an algorithm whose running time is $2^n$ on the same machine? [5 Marks]

**Q4)** Consider sorting $n$ numbers stored in array $A$ by first finding the smallest element of $A$ and exchanging it with the element in $A[1]$. Then find the second smallest element of $A$, and exchange it with $A[2]$. Continue in this manner for the first $n$ - 1 elements of $A$. Write pseudo-code for this algorithm, which is known as ***selection sort***. Why does it need to run for only the first $n$ - 1 elements, rather than for all $n$ elements? Give the best-case and worst-case running times of selection sort in Θ-notation. [5 Marks]

**Q5) a)** Suppose you are given $p$ sorted arrays, each with $m$ elements, and you want to combine them into a single array of $pm$ elements. Consider the following approach. Using the merge subroutine, you merge the first 2 arrays, then merge the 3*rd* given array with this merged version of the first two arrays, then merge the 4*th* given array with the merged version of the first three arrays, and so on until you merge in the final (*pth*) input array.

What is the running time taken by this successive merging algorithm, as a function of $p$ and $m$? [3 Marks]

**b)** What is a faster way to do the above merge procedure ? [5 Marks]

**Q6)** Consider the following pseudocode for calculating $x^y$ (where x and y are positive integers)

```
FindPower(x,y) :
        if y = 1
            return x
        otherwise
            c := x*x
            ans := FindPower(c,[y/2])
         if y is odd
            return x*ans
         otherwise return ans
```

what is overall asymptotic running time of the above algorithm (as a function of y)? [5 Marks]

**Q7)** Prove that T(n) is $\Theta(n^3)$ by finding appropriate constants. [5 Marks]

$$T(n) = \frac{1}{8}\, n^3 - 5n^2$$

**Q8)** Use a recursion tree to determine a good asymptotic upper bound on following recurrences. Please see Appendix of your text book for using harmonic and geometric series. (5 *5 = 25 Marks)

a) $T(n) = T(n/2) + O(n)^3$

b) $T(n) = 3T(4n/5) + \Theta(1)$

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

d) $T(n) = 2T(n/2) + n/\lg n$

e) $T(n) = 2T(n - 1) + \Theta(1)$