Problem Session 2

(Basics of algortihm analysis)

1. Order the following functions by growing rate. Show all your work.
   f1: $2^{\sqrt{n}}$, f2: 2^{n^2}, $f3$: e^{logn}, f4: $n^{1.6}$, f5: $n^{3.01}$, f6: $\log n^3$, f7:$(n+4)^{12}$, f8: $n^3 \log n$

   answer:
   f2 > f1 > f3 > f7 > f5 > f8 > f4 > f6

2. Give, using "big oh" notation, the worst case running times of the following procedures as a function of n. Make it as precise as possible and show all your work.

a. ans = $O(n^3)$

```
procedure matmpy ( n : integer);
      var
            i, j, k : integer;
      begin
            for I = 1 to n
                  for j = 1 to n do begin
                        C[I,j] = 0
                        for k = 1 to n do
                              C[I, j] = C[I,j] +A[I, k]*B[k,j]
                  end
      end
```

b. ans = $O(n^3)$

```
procedure mystery (n: integer)
      var
            i,j,k : integer;
      begin
            for i = 1 to n-1 do
                  for j= i+1 to n do
                        for k = 1 to j do
                              {some statement requiring O(1) time}
      end
```

c. ans = $O(n^2)$

```
procedure veryodd ( n : integer);
      var
            i, j, x, y : integer;
      begin
            for i = 1 to n do
                  if odd(i) then begin
                        for j = i to n do begin
                              x = x+1;
                        for j= 1 to i do
                              y = y + 1;
                  end
      end
```

d. ans = O(logn)

```
procedure mystery (n: integer)  //assuming n is a positive power of 2
      var
            i,j: integer;
      begin
            j = 0;
            x = 2;
            while x < n do begin
                  x = 2 * x;
                  j = j + 1;
            end
            writeln(j);
      end
```

3. Consider the following basic problem. You're given an array A consisting of n integers A[1], A[2],...., A[n]. You'd like to output a two-dimensional n-by-n array B in which B[i,j] (for i <j) contains the sum of array entries A[i] through A[j] - that is, the sum A[i] +A[i + 1] +....+ A[j]. (The value of array entry B[i,j] is left unspecified whenever i >=j, so it doesn't matter what is output for these values.)
Here's a simple algorithm to solve this problem.

For i=1, 2,...,n
    For j=i+1,...,n
        Add up array entries A[i] through A[j]
        Store the result in B[i,j]
    Endfor
Endfor

a. Do complexity anaysis.
b. Although the algorithm you analyzed is the most natural way to solve the problem - after all, it just iterates through the relevant entries of the array B, filling in a value for each – it contains some highly unnecessary sources of inefficiency. Give a different algorithm to solve this problem, with an asymptotically better running time.

Answer:

a. $O(n^3)$, $\Omega(n^3)$, $\Theta(n^3)$

b. complexity = $O(n^2)$

```
for i = 1 to n
      Set B[i,i+1] to A[i] + A[i+1]
for k = 2 to n-1
      for i = 1 to n-k
            Set j = i+k
            Set B[i,j] to be B[i,j-1] + A[j]
```