Advanced Algorithms and Data Structures
Algorithm Complexity
Exercises

Prepared by LY Rottana

1. Determine the Big-Oh *O* for the following code fragments:

a.

```
int sum = 0;
for (int i=0; i<n; i++)
     sum++;
```

b.

```
int sum = 0;
for (int i=0; i<n; i+=2)
     sum++;
```

c.

```
int sum = 0;
for (int i=0; i<n; i++)
     for (int j=0; j<n; j++)
          sum++;
```

d.

```
int sum = 0;
for (int i=0; i<n; i+=2)
     sum++;
for (int j=0; j<n; j++)
     sum++;
```

e.

```
int sum = 0;
for (int i=0; i<n; i++)
     for (int j=0; j<n*n; j++)
          sum++;
```

f.

```
int sum = 0;
for (int i=0; i<n; i++)
     for (int j=0; j<n*n; j++)
          for (int k=0; k<j; k++)
               sum++;
```

g.

```
int sum = 0;
for (int i=1; i<n; i=i*2)
     sum++;
```

h.

```
int a = 5;
int b = 10;
int c = a + b;
```

i.

```
int s = 0;
for (int i=0; i<100; i++){
     s += i;
}
```

j.

Assume that array A contains n values, **random** takes constant time, and **sort** takes $n \log n$ steps.

```
for (int i=0; i<n; i++){
     for (j=0; j<n; j++)
          A[j] = random(n);
     sort(A);
}
```

k.

```
int sum = 0;
if (n%2 == 0)
     for (int i=0; i<n; i++)
          sum++;
else
     sum = sum+n;
```

2. Determine the running time of the following code:

a. Addition of array members

```
int sum (int arr[], int n) {
     int i, total = 0;
     for (i=0; i<n; i++)
          total += arr[i];
     return total;
}
```

b. Addition of two matrix

```
void add (int a[][], int b[][], int c[][],int n)
{
     for (int i=0; i<n; i++)
          for (int j=0; j<n; j++)
               c[i][j] = a[i][j] + b[i][j];
}
```

c. Multiplication of two matrix

```
void mul (int a[][], int b[][], int c[][],int n)
{
     int i, j, k, sum;
     for (i=0; i<n; i++)
          for (j=0; j<n; j++) {
               sum = 0;
               for (k=0; k<n; k++)
                    sum = sum + a[i][k]*b[k][j];
               c[i][j] = sum;
}
```

3. We say that n_0 and c are witnesses to the fact that f(n) is O(g(n)) if for all $n \geq n_0$, it is true that f(n) $\leq$ c * g(n).
   a. If $n_0 = 1$, what is the smallest value of c such that n_0 and c are witnesses to the fact that $(n + 2)^2$ is $O(n^2)$?
   b. If $c = 5$, what is the smallest non-negative integer n_0 such that n_0 and c are witnesses to the fact that $(n + 2)^2$ is $O(n^2)$?
   c. If $n_0 = 0$, for what values of c are n_0 and c witnesses to the fact that $(n + 2)^2$ is $O(n^2)$?

4. Algorithms A and B spend exactly $T_A(n) = 0.1n^2 \log_{10}n$ and $T_B(n)$=$2.5n^2$ microseconds, respectively, for a problem of size n. Choose the algorithm, which is better in the Big-Oh sense, and find out a problem size $n0$ such that for any larger size $n > n_0$ the chosen algorithm outperforms the other. If your problems are of size $n \leq 10^9$, which algorithm will you recommend to use?

5. Given an algorithm to search for a number in an array A of n elements shown below, analyze this algorithm to find the complexity (big O) of its best case, worst case, and average case.

```
for i ← 0 to n-1 do
     if (A[i]=val) then
          return i
return -1
```

6. Given an array A storing n integers (random values), write an algorithm that inverses the order of the array. Determine the Big-Oh for your algorithm.