Gebreegziabher Redda

#1

```
/** Returns the sum of the integers in given array. */
public static int example1(int[ ] arr) {
int n = arr.length, total = 0;
for (int j=0; j < n; j++) // loop from 0 to n-1
total += arr[j];
return total;
}
```

Cost	Time
C1	1
C2	1
C3	n+1
C4	n
C5	1

Total =1+1+n+1+n+1

But as n growths larger the constant will negligent so their order will be O(n)

So that the time required for this algorithm is proportional to n

#2

```
/** Returns the sum of the integers with even index in given array. */
public static int example2(int[ ] arr) {
int n = arr.length, total = 0;
for (int j=0; j < n; j += 2) // note the increment of 2
total += arr[j];
return total;
}
```

Cost	**Time**
C1	**1**
C2	**1**
C3	**n/2+1**
C4	**n/2**
C5	**1**

Total =1+1+n/2+1+n/2+1+1

But as n growths larger the constant will negligent so their order will be O(n)

So that the time required for this algorithm is proportional to n

#3

```
/** Returns the sum of the prefix sums of given array. */
public static int example3(int[ ] arr) {
int n = arr.length, total = 0;
for (int j=0; j < n; j++) // loop from 0 to n-1
for (int k=0; k <= j; k++) // loop from 0 to j
total += arr[j];
return total;
}
```

Cost	Time
C1	1
C2	1
C3	n
C4	(n^2)/2 +n/2
C5	(n^2)/2 +n/2
C6	1

Total =1+1+n+((n^2)/2 +n/2) + ((n^2)/2 +n/2) +1

But as n growths larger the constant will negligent so their order will be O(n^2)
So that the time required for this algorithm is proportional to n^2

#4

```
/** Returns the sum of the prefix sums of given array. */
public static int example4(int[ ] arr) {
int n = arr.length, prefix = 0, total = 0;
for (int j=0; j < n; j++) { // loop from 0 to n-1
prefix += arr[j];
total += prefix;
}
return total;
}
```

Cost	**Time**
C1	**1**
C2	**1**
C3	**1**
C4	**n+1**
C5	**n**
C6	**n**
C7.	**1**

Total =1+1+1+n+1+n+n+n+1

But as n growths larger the constant will negligent so their order will be O(n)
So that the time required for this algorithm is proportional to n

#5

```
/** Returns the number of times second array stores sum of prefix sums from
rst. */
public static int example5(int[ ] first, int[ ] second) { // assume equal-
ngth arrays
int n = first.length, count = 0;
for (int i=0; i < n; i++) { // loop from 0 to n-1
int total = 0;
for (int j=0; j < n; j++) // loop from 0 to n-1
for (int k=0; k <= j; k++) // loop from 0 to j
total += first[k];
if (second[i] == total) count++;
} 46 return count;
}
```

Cost	Time
C1	1
C2	1
C3.	n+1
C4	n
C5	n+1
C6	n^2
C7	n^3/2+n^2/2
C8	n^3/2+n^2/2
C9	n
C10	n
C11	1

Total =1+1+n+1 +n+1+n^2+n^3/2+n^2/2+ n+ n^3/2+n^2/2+n+1

But as n growths larger the constant will negligent so their order will be O(n^3)

So that the time required for this algorithm is proportional to n^3

#6

```
public class BigO {
 /*   6- Write a function for each of the following growth rates:
            methode1- O(n)*/
      int methode1(){
          int a=0;
          int n=100;
          for(int i = 0; i<n; i++){
              a+=i;
          }
       return a;
    }
    /*methode2- O(n log n) */
      int  method2(){
          int x=0;
          int n=100;
          for(int i = 0; i<n; i++){
              for(int j=n;j>0;j=j/2)
                  x+=i*j;
          }
          return x;
      }
```

```
        /* methode3- O(n3)*/
        int  method3(){
            int y=0;
            int n=100;
            for(int i = 0; i<n; i++){
                for(int j=0;j<n;j++)
                    for(int k=0;k<j;k++)
                    y+=i*j;
            }
            return y;
        }
}
```

#7 . We have two case the best case and worst-case analysis to measure time taken to computes the algorithms. The best case is the minimum time taken that an algorithm requires to solve problems, but this case will not consider since it is minimum time it may not affect for our systems efficiency, but our main focus is worst case analysis because this means the maximum time taken that an algorithm require to solve problems because it will help us to manage our systems efficiency. If we know the worst case, we can be able to handle how fast the algorithm will perform this leads us to manage our algorithm efficiency.