CPSC 319 Assignment 1: Analysis of Algorithm

Name:Shanzi Ye

Instructor:Bajwa, Sohaib

Submission Date: July,10,2022

Question 1: (3 marks)

Consider an algorithm where exact number of steps has been defined as $f(n)=4n+3n^2-1$. Perform a growth rate analysis by identifying Time Unit, Prop to and rate. Below is general template we discussed in the class:

It requires $4n+3n^2-1$ time units to solve a problem of size n

It requires time proportional to $3n^2$

Its growth rate is Quadratic

Question 2: (2 marks)

Consider the below functions we discussed in our lecture:

Linear, logarithmic, linear logarithmic exponential, quadratic, constant, cubic,

Write the above function from top to bottom order from most to least efficient considering the input size becomes very large.

Solution:

1.	**Constant** →	**Most efficient**
2.	**Logarithmic**	
3.	**Linear**	
4.	**Linear logarithmic**	
5.	**Quadratic**	
6.	**Cubic**	
7.	**Exponential** →	**Least efficient**

Question 3: (5 marks)

Consider the below code fragment where n is the size of the input:

```
int test = 0;
  for (int i = 0; i < n; i++){
    for (int j = 0; j < n; j++){
      test = test + i * j;
    }
  }
```

Solution:

1	Line #	Algorithm	Cost	Times	Comments
2	1	int test = 0;	C1=1	1	one op: assign
3	2	for (int i = 0; i < n; i++){	C2=1	1	Loop initializing (assigning a value)
4			C3=2	n	Loop incrementation: • Two ops: an addition and an assignment • done n times
5			C4=1	n+1	Loop termination test: • a comparison i < n each time • n successes and one failure
6	3	for (int j = 0; j < n; j++){	C5=1	1	Loop initializing (assigning a value)
7			C6=2	n*n	Loop incrementation: • Two ops: an addition and an assignment • done n*n times
8			C7=1	n*(n+1)	Loop termination test: • a comparison j < n each time • n successes and one failure
9	4	test = test + i * j;	C8=3	n*n	Three ops per iteration (i.e., mult, add,assign). • executed n*n times

f(n) =C1*1+C2*1+C3*n+C4*(n+1)+C5*1+C6*(n^2)+C7*(n(n+1))+C8*(n^2)

= C1+C2+C3*n+C4+C4*n+C5+C6*(n^2)+C7*(n^2)+C7*n+C8*(n^2)

=(C6+C7+C8)*(n^2)+(C3+C4+C7)*n+(C1+C2+C4+C5)

=(2+1+3)*(n^2)+(2+1+1)*n+1+1+1+1

=6*(n^2)+4n+4

Therefore,its growth rate is quadratic and it Big-O running time is O(n^2).

Question 4: (2 marks)

Consider the below code fragment:

```
int func(){
   int test = 0;
   for (int i = 0; i < n; i++){
      test = test + 1;
   }
   for (int j = 0; j < n; j++){
      test = test - 1;
   }
   return 0;
}
```

What is its Big-O running time? Explain your answer.

Solution:

Its Big-O running time is O(n).

This question is different from the last question. Question 3 is a nested loop so the big-O running time is O(n)*O(n) = O(n^2).

Instead, this question contains two independent "for loops" and each loop has nothing to do with another. Each loop's Big-O running time is O(n). (Note: O(n) not O(log n) because of i++ instead of i*2)

Since the two loops are independent, we just need to add them up. The final Big-O Running time should be O(n)+O(n) = O(n).

Question 5: (5 marks)

Consider the below code fragment:

```
int func(n){
  int i = n;
  int count = 0;
  while (i > 0){
    count = count + 1;
    i = i // 2;
  }
  return 0;
}
```

What is the growth rate function? Also identify the Big-O running time? Explain your answer.

Solution:

Line #	Algorithm	Cost	Times	Comments
2	int i = n;	C1=1	1	one op: assign
3	int count = 0;	C2=1	1	one op: assign
4	while (i > 0){	C3=1	log(n)+2	Loop termination test: • a comparison i >0 each time • log(n)+1 successes and one failure
5	count = count + 1;	C4=2	log(n)+1	Two ops per iteration (i.e. add, assign). • executed log(n)+1 times
6	i=i/2;	C5=2	log(n)+1	Two ops per iteration (i.e. division, assign). • executed log(n)+1 times
8	return 0;	C6=1	1	one op:return from a method

$$F(n)= C1*1+C2*1+C3*(\log(n)+2)+C4*(\log(n)+1)+C5*(\log(n)+1)+C6*1$$

$$=C1+C2+C3*\log(n)+C3*2+C4*\log(n)+C4+C5*\log(n)+C5+C6$$

$$=\log(n)(C3+C4+C5)+C1+C2+C3*2+C4+C5+C6$$

$$=\log(n)(1+1+2)+1+1+1*2+2+2+1$$

$$=4\log(n)+9$$

The growth rate function is 5log(n)+9. In order to find the Big-O running time,we just need to find the fastest growing term. Therefore,the Big-O running time is O(log n)

Question 6: Write a scenario (or a code fragment), whose complexity is O(n^3) **(2 marks)**

Solution:

```
int test = 0;
for (int i = 0; i < n; i++){
    for (int j = 0; j < n; j++){
        for (int y = 0;y<n;y++) {
            test = test + i * j;
        }
    }
}
```

Question 7: If an algorithm performing at O(n^2) has the integer 10 as input, what is the worst case scenario for the algorithm? **(1 marks)**

Solution:

The worst scenario occurs when n equals 10. In other words, the worst scenario occurs when the algorithm runs 100 times.(Note:10^2=100)

Question 8: Use Big O Notation to describe the time complexity of the following function that determines whether a given year is a leap year: **(1 marks)**

```
bool foo(temp) {
    return (temp % 100 == 0) ? (temp % 400 == 0) : (temp % 4 == 0);
}
```

Solution:

The Big-O running time is O(1)

Question 9: Use Big O, to describe the time complexity of this function, which is below: **(4 marks)**

```
int chessboardSpace(numberOfGrains)
 {  chessboardSpaces = 1;
      placedGrains = 1;
      while (placedGrains < numberOfGrains) {
           placedGrains *= 2;
           chessboardSpaces += 1;
}
return chessboardSpaces; }
```

Explain your answer.

Solution:

Let we assume numberofgrain is 16.

PlacedGrains will be

1 (for the first time) (both in Loop termination test and while loop execution)

2 (for the second time) (both in Loop termination test and while loop execution)

4 (for the third time) (both in Loop termination test and while loop execution)

8 (for the fourth time) (both in Loop termination test and while loop execution)

16 (for the fifth time) (only will be used in loop termination test)

Then we can get Placedgrains = 2^0, 2^1, 2^2, 2^3, 2^4.

Thus we can conclude that Placedgrain = 2^0, 2^1, 2^2, 2^3, 2^4,........2^k.

Therefore 2^k = numberofgrain

K=log(n)

	A	B	C	D	E
1	Line	Algorithm	Cost	Times	Comments
2	Line2	{ chessboardSpaces = 1;	C1=1	1	one op: assign
3	Line3	placedGrains = 1;	C2=1	1	one op: assign
4	Line4	while (placedGrains < numberOfGrains) {	C3=1	log(n)+1	Loop termination test: • a comparison placedGrains < numberOfGrains each time • log(n) successes and one failure
5	Line5	placedGrains *= 2;	C4=2	log(n)	Two ops per iteration (i.e. mult, assign). • executed log(n) times
6	Line6	chessboardSpaces += 1;	C5=2	log(n)	Two ops per iteration (i.e. add, assign). • executed log(n)+1 times
7	Line8	return chessboardSpaces; }	C6=1	1	one op:return from a method

$$F(n)=C1*1+C2*1+C3*(\log(n)+1)+C4*\log(n)+C5*\log(n)+C6*1$$
$$=C1+C2+C3*\log(n)+C3+C4*\log(n)+C5*\log(n)+C6$$
$$=\log(n)(C3+C4+C5)+C1+C2+c3+C6$$
$$=\log(n)(1+2+2)+1+1+1+1$$
$$=5\log(n)+4$$

The growth rate function is 5log(n)+4. In order to find the Big-O running time,we just need to find the fastest growing term and take the coefficient.Therefore,the Big-O running time is O(log n)