HW1 DATA STRUCTURE

Name: Shahad Abdullah Almuhizi

ID: 436201525

Section: 44126

Problem 1:

1)

Choose k = 1.
Assuming n > 1, then

$$f(n)/g(n) = (5n^2 + n + 1)/n^2 < (5n^2 + n^2 + n^2)/n^2 = 8$$

choose C=8.
Note that $2n < 2n^2$ **and** $1 < n^2$**.**
Thus, $5n^2 + 2n + 1$ **is O(** n^2 **).**
Because $5n^2 + 2n + 1 <= 8n^2$ **whenever** $n >= 1$**.**
5(1)+2(1)+1<=8
8<=8
Another solution:

$$5n^2 + 2n + 1 <= 5n^2 + 2n^2 + 1^2 = 8n^2$$

n_0 = 1
c=8

2)

The Big O is O(n^2).
proof:

$$n^2 + nlog(n) <= c \,.\, n2$$

$$n^2 + nlog(n) <= n^3 + n^2log(n)$$

$$n^2 + nlog(n) <= n^2(1 + log(n))$$

n_0 =2
c=1
6<=8
The Big O is O(n^2).

3)

$f(n)/g(n) = (2n^3)/n^2 > (2n^3 - 1)/n^2 > (2n^3 - n^2)/n^2$

=2n-1
n>2c+1 ==> 2n-1>c
and f(n)>cn
choosing n>1,n>k, and n>2c+1
implies n>k and f(n)>cn.
Therefore, $2n^3$ **is not O(**n^2**).**

another solution:

$n^3 <= c \cdot n^2$

$n <= c$

the inequality cannot be satisfied since c must be a constant.

4)

a)

The dominant term with the steepest increase in n is 0.1n^2

b) The lowest Big-Oh complexity of the algorithm is O(n^2 **).**

5)

a) True.

b) False, $100n^3 + 8n^2 + 5n = O(n^3)$

6)

As a,b>0, then a and b could be the same constant, which does not affect the Big O, so this states that log$_a$(n) is an element from O(log$_b$(n)).

7)

$a^n >= c \cdot b^n$ for every a>b>0
$(a/b)^n >= c$ for every a>b>0
$n \cdot log(a/b) >= log(c)/log(a/b)$ for every a>b>0
$n >= log(b \cdot c/a)$ for every a>b>0
For every $n >= log(b \cdot c/a), a^n >= b^n$
Therefore, a^n is not $O(b^n)$.

Based on the condition, as a>b>0, then a^n is bigger than b^n , which makes it impossible for $a^n \in O(b^n)$, because their value has an affect on the power n, and that means a^n is not an element from $O(b^n)$.

Problem 2:
1)

	Statements	S/E	Freq.	Total
1	int sum = 0;	1	1	1
2	for (int i = 1; i <= n ; i++)	1	n+1	n+1
3	for(intj=0;j< 2*i;j++)	1	n(n+1)+n	$n^2 + 2n$
4	sum += j;	1	n(n+1)	$n^2 + n$
5	return sum;	1	1	1
Total Operations				$2n^2 + 4n + 3$
Big O				$O(n^2)$

2)

	Statements	S/E	Freq.	Total
1	for (int i = 0; i < n * n * n; i++) {	1	$n^3 + 1$	$n^3 + 1$
2	System.out.println(i);	1	n^3	n^3
3	for (int j = 2; j < n; j++)	1	n^3(n-1)	$n^4 - n^3$
4	System.out.println(j); }	1	n^3(n-2)	$n^4 - 2n^3$
5	System.out.println("End!");	1	1	1
Total Operations				$2n^4 - n^3 + 2$
Big O				$O(n^4)$

3)

	Statements	S/E	Freq.	Total
1	int k = 100, sum = 0;	1	1	1
2	for (int i = 0; i < n; i++)	1	n+1	n+1
3	for (j = 1; j <= k; j++) {	1	n(101)	101n
4	sum = i + j;	1	n(100)	100n
5	System.out.println(sum);	1	n(100)	100n
6	}	0	0	0
Total Operations				302n+2
Big O				$O(n)$

Problem 3:

1) Since the algorithm chooses log n elements, and each element needs O(n) time calculation,
so the worst case is: O(n log n), where log n represent the number of elements, and n represent the time calculation for each element.

2)

The worst case when all array elements are even, which is O(n2), where n represent the number of even elements, and the other n represent the time calculation for each even number. The best case is when all array elements are odd, which is O(n log n) where n represents the number of odd elements and the log n represent the time calculation for each odd element.

3)

a) O(n)
b) O(n)
c) O(mn)
d) O(mn)
e) O(mkn)

Problem 4:
Best case n= 0
worst case n=10000

best case analysis:

	Statements	S/E	Freq.	Total
1	int func1(int[] A, int n) {	0	0	0
2	int i =0;	1	1	1
3	int j = n - 1;	1	1	1
4	int sum = 0;	1	1	1
5	while (i <=j) {	1	1	1
6	if(A[i] >A[j]){	1	0	0
7	for (int k = i; k <= j; k++) {	1	0	0
8	sum += A[k];	1	0	0
9	}	0	0	0
10	}	0	0	0
11	i++;	1	0	0
12	j--;	1	0	0
13	}	0	0	0
14	return sum;	1	1	1
15	}	0	0	0
Total Operations				5
Big O				O(1)

worst case analysis:

	Statements	S/E	Freq.	Total
1	int func1(int[] A, int n) {	0	0	0
2	int i =0;	1	1	1
3	int j = n - 1;	1	1	1
4	int sum = 0;	1	1	1
5	while (i <=j) {	1	10001	10001
6	if(A[i] >A[j]){	1	10000	10000
7	for (int k = i; k <= j; k++) {	1	10000(10001)	10000(10001)
8	sum += A[k];	1	10000(10000)	10000(10000)
9	}	0	0	0
10	}	0	0	0
11	i++;	1	10000	10000
12	j--;	1	10000	10000
13	}	0	0	0
14	return sum;	1	1	1
15	}	0	0	0
Total Operations				200050005
Big O				O(1)

Problem 5:

1) This function takes O(n) in space, because the creation of an array takes n in space complexity.

2) This function takes $O(n^2)$in space. because the two dimensional array takes $O(n^2)$in space.

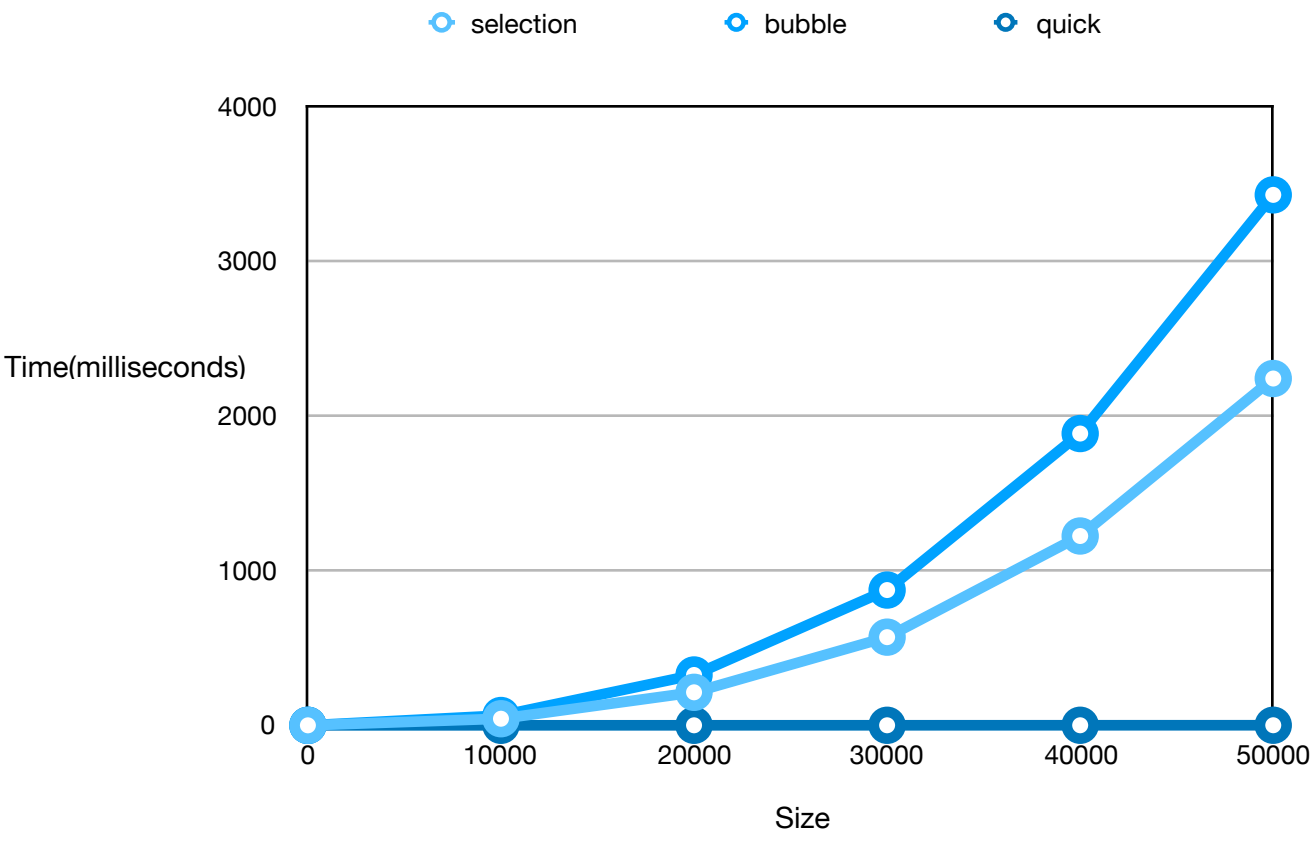
Problem 6:

1)

```
1 public class Test {
2 public static void main (String [] args){
3 long selection=0;
4 long bubble=0;
5 long quick=0;
6 long time1=0;
7 long time2=0;
8 for (int i=10000; i<= 50000; i=i+10000){
9 double[] selectionSort= new double[i];
10 double[] bubbleSort= new double [i];
11 double[] quickSort= new double [i];
12 for(int j=0; j<i; j++){
13 selectionSort[j]=bubbleSort[j]=quickSort[j]=Math.random(); }
14 for (int x=0 ; x<100 ; x++){
15 time1=System.nanoTime();
16 Sort.selectionSort(selectionSort,i);
17 time2= System.nanoTime();
18 selection= selection+(time2-time1);
19 time1=System.nanoTime();
20 Sort.bubbleSort(bubbleSort,i);
21 time2= System.nanoTime();
22 bubble= bubble+(time2-time1);
23 time1=System.nanoTime();
24 Sort.quickSort(quickSort,i);
25 time2= System.nanoTime();
26 quick= quick+(time2-time1);}
27 System.out.println(“Average of selection sort with size:"+ i+ " is: "+ ((double)selection/100)/1000000);
28 System.out.println("Average of bubble sort with size:"+ i+ " is: "+ ((double)bubble/100)/1000000);
29 System.out.println("Average of quick sort with size:"+ i+ " is: "+ ((double)quick/100)/1000000);
30 }}}
```

2)

Size/sort type	Selection sort	Bubble sort	Quick sort
10000	42.628497859999996	63.469121380000004	0.12743546
20000	213.31163769	325.38924303	0.21158648000000002
30000	569.6897540800001	874.02780991	0.31498278
40000	1223.21725969	1885.20172809	0.42093293
50000	2240.7154903	3427.23297256	0.54454113



3) Quick sort is the fastest since it takes the shortest time to sort the array.

4) Selection sort is faster since it takes shorter time to sort the array with different sizes, bubble sort has larger growth rate than the selection sort because bubble sort take longer time to sort the array, even they have the same time complexity with big O(n^2).