Problem 1

Note :
$5n^2 + 2n^2 + 1n^2 \geq cn^2$
لن يؤثر التغير في المتراجحة ولكن سيسهل علينا إيجاد الثابت والقسمة وهي إحدى طرق الإثبات المعترف بها

1. **Show that $5n^2 + 2n + 1$ is O(n^2)**

   $5n^2 + 2n + 1 \leq cn^2$

   $5n^2 + 2n + 1 \leq 5n^2 + 2n^2 + 1n^2$

   $n^2 + 2n + 1 \leq 8n^2$, **c= 8**

   For C=8, $n_0 = 1$.

   $8 \leq 8$

   For all $n \geq n_0$ the statment will be true so is $O(n^2)$

2. **What is the Big oh of $n^2 + nlog(n)$? prove your answer.**

   F(n) is O(n^2)

   $n^2 + n\log(n) \leq cn^2$

   $n^2 + n\log(n) \leq n^2 + n^2$ $(n^2 > nlogn)$ *it wiil not effectly wrong*

   $n^2 + n\log(n) \leq 2n^2$ **c= 2**

   For C=2, $n_0 = 1$.

   $2 \leq 2$

   For all $n \geq n_0$ the statment will be true so is $O(n^2)$

3. **Show that $2n^3 \notin$ O(n^2).**

   $2n^3 \leq cn^2$

   $\frac{2n^3}{c} \leq n^2$

   let c=2 we will get $n^3 \leq n^2$ which is false for any value of n such that n > 1.

   ***

   Another soln :

   $2n^3 \leq cn^2$

   $\frac{2n^3}{n^2} \leq c$

   $2n \leq c$

   Where c is a constant so it will not change

   But n is a variable so it will take any number that make the sentence false

4. **Assume that the expression below gives the processing time f(n) spent by an algorithm for solving a problem of size n.**

   $$10n + 0.1n^2$$

   (a) **Select the dominant term(s) having the steepest increase in n.**

   The terms which has the most degree ($0.1n^2$)

   (b) **Specify the lowest Big-Oh complexity of the algorithm.**

   F(n) is $O(n^2)$

5. Determine whether each statement is true or false and correct the expression in the latter case:

(a) 100n^3 + 8n 2 + 5n is O(n 4). True.

(b) 100n^3 + 8n 2 + 5n is O(n 2 log n). False, $n^2 logn\ not\ \leq n^3$. the correct answer is O(n^3) which is the lowest big-oh .

6. **Show that** $\log_a n$ **∈ O(**$\log_b n$ **)) for all a, b > 0.**

$\log_a n \leq C \log_b n$

Let:

$\log_a n = x$, $\log_b n = y$

$a^x = n$, $b^y = n$

$logn = xloga$ ---------- logn= $y\ logb$

$xloga = ylogb$

$\log_a n\ log\ a = \log_a n\ logb$

$\log_a n = \frac{\log b}{\log a} \log_b n$ let C=$\frac{\log b}{\log a}$

$\log_a n = c\ \log_b n$

F(n) is O($\log_b n$)

So $\log_a n \in O(\log_b n)$

7. **Show that** a^n **∈/ O(**b^n **)if a > b > 0**

$a^n \leq cb^n$

$nlog\ a \leq logC + n \log b$

$n(\log a - \log b) \leq logC$

Log c is a **constant it** will not change

Right hand side n is **a variable** so there is a number will make the statement false

Let c= 2 , a=8 , b=4, $n_{0=1}$ $n = 3$

2(log 8 -log 4) ≤ log 2

2(3-2) ≤ 1

2≤ 1 false

Problem 2

Analyze the following code excerpts:

1)

No	Statements	S/E	Frequency	Total
1	`int sum = 0;`	1	1	1
2	`for ( int i = 1; i <= n ; i ++)`	1	$n+1$	$n+1$
3	`for ( int j = 0; j < 2* i ; j ++)`	1	$n(n+1)+n$	n^2+2n
4	`sum += j ;`	**1**	$\boldsymbol{n(n+1)}$	$\boldsymbol{n^2+n}$
5	`return sum ;`	1	1	1
Total operation		$2n^2+4n+3$		
Big oh		$O(n^2)$		

2)

No	Statements	S/E	Frequency	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 operation		$2n^4-n^3+2$		
Big oh		$O(n^4)$		

3)

No	Statements	S/E	Frequency	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	$101n$	$101n$
4	`sum = i + j ;`	1	$100n$	$100n$
5	`System.out.println ( sum ) ; }`	1	$100n$	$100n$
Total operation		$302n+2$		
Big oh		$O(n)$		

Problem 3

1. **Given an n-element array X, Algorithm B chooses log n elements in X at random and executes an O(n)-time calculation for each. What is the worst-case running time of Algorithm B?** (Question R-4.30 page 184 of the textbook)

   worst-case running time = number of inputs * big oh

   :. sine we have $logn$ elements, and each element needs O(n)-time calculation

   then the worst-case running time is: **O $(nlogn)$**

2. **Given an n-element array X of integers, Algorithm C executes an O(n)-time computation for each even number in X, and an O(log n)-time computation for each odd number in X. What are the best-case and worst-case running times of Algorithm C?** (Question R-4.31 page 184 of the textbook)

   - The best-case running time when all of the array elements are odd number,

   O $(nlogn)$.

   - The worest-case running time when all of the array elements are even number,

   O (n^2).

3. Give in asymptotic notation the running time for the following algorithms:

(a) Vector-vector addition (the vectors are of size n). **O(n)**

(b) Dot product of two vectors (the vectors are of size n). **O(n)**

(c) Matrix-vector multiplication (the matrix is of size m × n, the vector is of size n). **O(nm)**

(d) Matrix addition (the two matrices are of size m × n). **O(nm)**

(e) Matrix-Matrix multiplication (the two matrices are of size m × k and k × n respectively). **O(nmk)**

Problem 4

For the following function:

1. **Give two example inputs leading to the best and worst running time respectively.**

- The best-case running time when the array elements are ordered BY (increasing) order, because the over flow of executing will skip parts of the code, For example:

N=4, A= {4,5,6,7}

- The worst-case running time is happened when the array elements are ordered in decreasing order, because every statement in the code will execute, For example:

N=3, A ={9,8,7}

2. **Analyze the performance of the function in each case (best and worst).**

-Best case performance:

No	Statements	S/E	Frequency	Total
1	`int func1 ( int [] A , int n ) {`	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	(N+1\2)+1	(N+1\2)+1
6	`if ( A [ i ] > A [ j ]) {`	1	(N+1\2)	(N+1\2)
7	`for ( in t k = i ; k <= j ; k ++){`	1	0	0
8	`sum += A [ k ]; } }`	1	0	0
9	`i ++;`	1	(N+1\2)	(N+1\2)
10	`j - -; }`	1	(N+1\2)	(N+1\2)
11	`return sum ; }`	1	1	1
Total operation			2n+7	
Big oh			O(n)	

-worst case performance:

No	Statements	S/E	Frequency	Total
1	`int func1 ( int [] A , int n ) {`	0		
2	`int i = 0;`	1		
3	`int j = n - 1;`	1		
4	`int sum = 0;`	1		
5	`while ( i <= j ) {`	1		
6	`if ( A [ i ] > A [ j ]) {`	1		
7	`for ( in t k = i ; k <= j ; k ++){`	1		
8		1		
9	`sum += A [ k1 ]; } }`	1		
10		1		

11	`i ++;`
	`j - -; }`
	
	`return sum ; }`
		1		

Total operation				
Big oh				

Problem 5

1)

```
int sum1 ( int [] A , int n ) {
int sum = 0;
for ( int i = 0; i < n ; i ++) {
sum+= A [ i ];}
 return sum ;}
```

2 spaces

1 space

1 space

Total spaces = 4

In the above code:

The parameters count as one for each.

`int sum = 0; int i = 0 ; the memory will reserve a` 1 space for each primitive variable .

`A[i] is not count because it has reserved space before Here we just count the spaces that reserved in actually algorithm`

the space complexity of this function is: $O(1)$

2) the function sum2 is O(n) in space (why?)

```
int sum2 ( int [] A, int n ) { 2 spaces
int sum = 0; 1 space
for (int i = 0; i < n ; i ++) { 1 space
int [] B = new int [ i + 1]; n space
for ( int j = i ; j <= i ; j ++) {1 space
B[ j ] = A [ j ] - A [ i ]; }
for ( int j = i ; j <= i ; j ++) {1 space
sum += B [ j ]; } }
return sum ; }
```

Total spaces = $n + 6$

In the above code:
The parameters count as one for each.
Premtive data type `int sum = 0; int i = 0; int j = I;` the memory will reserve a 1 space for each and n space for the array B (`int [] B`).

3) What is the space complexity of the following function? Justify your answer.

```
void func3 ( in t [] A, in t n) { 2 spaces
in t [][] B = new in t [ n ][];
for (int i = 0; i < n ; i ++) { 1space
B[ i ] = new in t [ i + 1]; n(n+1)/2 space (number of column depend
on the index of row )
for (int j = 0; j <= i ; j ++) { 1space
if ( A [ j ] > A [ i ])
B[ i ][ j ] = A [ j ];
else
B[i][j] = 0; } } }
```

total: $((n^2 + n)\backslash 2) + 4$

*** Note that : each row has different number of column

The parameters count as one for each.
the memory will reserve a 1 space for each primitive variable (`int i = 0; int j = 0; `) , but for the array B (`in t [][] B = new in t [ n ][];`) ;
the memory will reserve $\frac{n(n+1)}{2}$ space in memory .
From the code we can conclude that the first row in the Array has one column and the second row has two columns and the third has three columns ...
Summation law ($\frac{n(n+1)}{2}$).
the space complexity of this function is: $O(n^2)$

POBLEM 6:

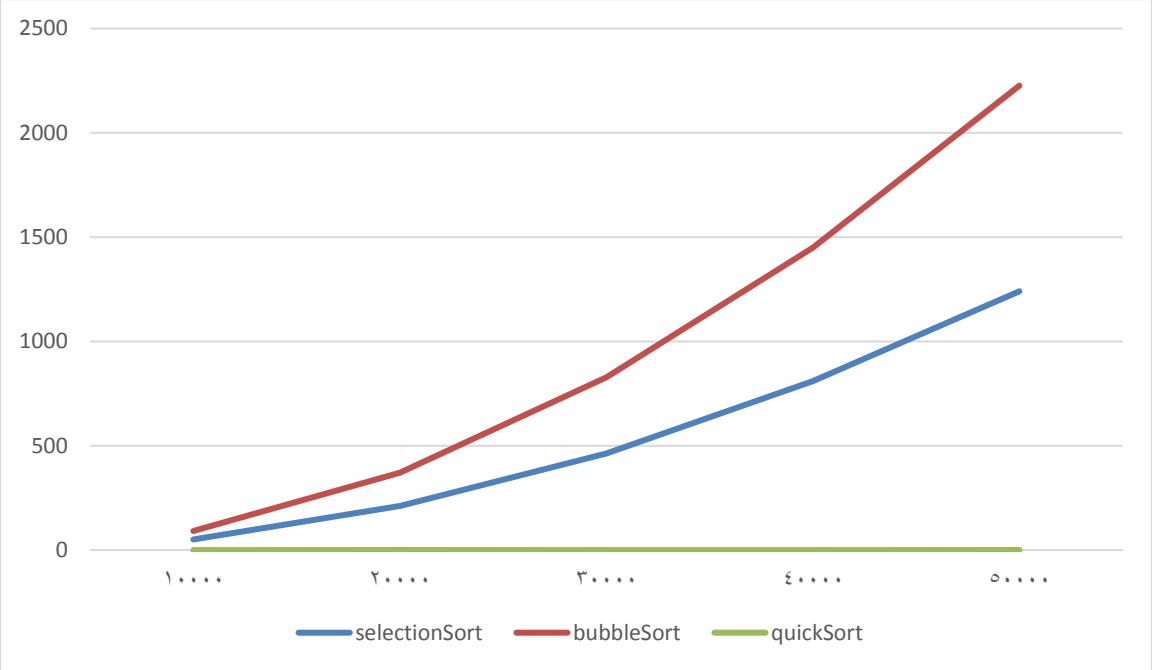
1-

```
1 import java.util.Arrays;
 2 public class test{
 3 public static void main (String []args) {
 4 double times=0, timeb=0, timeq=0;
 5 for(int i=10000; i<=50000; i+=10000){
 6    double [] lists = new double [i];
 7      double [] listb = new double [i];
 8   double [] listq = new double [i];
 9 times=0; timeb=0; timeq=0;
10    for(int j=0 ; j<i; j++)
11       lists[j]=listb[j]=listq[j]=Math.random();
12
13 for(int j=0; j<100; j++){
14 double before=System.nanoTime();//calculate time before
calling
15 Sort.selectionSort(lists,i);
16 double after=System.nanoTime();//calculate all time from
begning until finish the method
17 times+=(after-before);//the diffrence between them will be
the actual time of the method
18
19 before=System.nanoTime();
20 Sort.bubbleSort(listb,i);
21 after=System.nanoTime();
22 timeb+=(after-before);//actual time
23
24 before=System.nanoTime();
25 Sort.quickSort(listq ,i);
26 after=System.nanoTime();
27 timeq+=(after-before);
28 }
29 System.out.println("the info for the array of size:
"+i+"\n");
30 System.out.println("Average of
selection:"+(times/100)/1000000);
31 System.out.println("Average of
bubble:"+(timeb/100)/1000000);
32 System.out.println("Average of
quick:"+(timeq/100)/1000000+"\n");

35 }
36 }
37 }
```

2-

Size	selectionSort	bubbleSort	quickSort
10000	50.61981836	91.35688363	0.16296443
20000	211.50464313	370.37050436000004	0.11534798
30000	462.42521143	827.40160633	0.14929496
40000	808.80823213	1449.60451333	0.14593328
50000	1240.36412299	2226.2889896300003	0.16837926



3- Which of the three algorithms is the fastest?

Quicksort is the fastest since it takes the shortest time to sort the array.

4-Which of selection sort and bubble sort is faster? Which one has a larger growth rate?
Selection sort is faster since it takes shorter time to sort the array with different sizes, bubble sort has the largest growth rate .