HomeWork 1

# Problem1:

## 1-

|  |  |  |
| --- | --- | --- |
| function | Big oh | order |
| 4n log n + 2n | O (n log n) | 6 |
| 210 | O (1) | 1 |
| 2log n = n log2 = n | O (n) | 2 |
| 3n+100log n | O (n) | 3 |
| 4n | O (n) | 4 |
| 2n | O (2n) | 9 |
| n2+10n | O (n2) | 7 |
| n3 | O (n3‎) | 8 |
| n log n | O (n log n) | 5 |

## 2-

## 

## ]

## 

## 3-

## 4-

From Logarithm rules: log (ac) = log (a) + log (c)

Which is true for c=1 when n0 =1

# 5-

## 

Which is true for c=15 when n0 = 1

### Another solution:

For c=15 when n0 =1

# Problem2:

## 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.

## 3-

Since algorithm D calls algorithm E for each element in the array, the time computation will be the summation of which gave us the worst case with **O(n2).**

# Problem3:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | statements | S/E | freq | total |
| 1 | public void func1 ( int n) {  for (int i = 0; i < n \* log(n); i++) {  System . out . println (i);  for (int j = 2; j < n; j++) {  System .out . println (j);  }  }  System .out . println (" Goodbye !");  } | 0 | 0 | 0 |
| 2 | 1 | nlogn+1 | nlogn+1 |
| 3 | 1 | nlogn | nlogn |
| 4 | 1 | nlogn(n-2+1) | n2logn-nlogn |
| 5 | 1 | nlogn(n-2) | n2logn-2nlogn |
| 6 | 0 | 0 | 0 |
| 7 | 0 | 0 | 0 |
| 8 | 1 | 1 | 1 |
| 9 | 0 | 0 | 0 |
|  | Total operation | 2n2logn-nlogn+2 | | |
|  | Big oh | O(n2logn) | | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | statements | S/E | freq | total |
| 1 | public void func2 ( int n) {  for (int i = 0; i < n \* n; i++) {  System . out . println (i);  for (int j = 2 \* n; j > n; j --) {  System .out . println (j);  }  }  System . out . println (" Goodbye !");  } | 0 | 0 | 0 |
| 2 | 1 | n2-0+1 | n2+1‎ |
| 3 | 1 | n2 | n2 |
| 4 | 1 | n2(2n—n+1) | 2n3-n3+n2 = n3+n2 |
| 5 | 1 | n2(2n-n) | 2n3-n3= n3 |
| 6 | 0 | 0 | 0 |
| 7 | 0 | 0 | 0 |
| 8 | 1 | 1 | 1 |
| 9 | 0 | 0 | 0 |
|  | Total operation | 2n3+3n2+2 | | |
|  | Big oh | O(n3) | | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | statements | S/E | freq | total |
| 1 | public void func3 ( int n) {  for (int i = n; i > 0; i --) {  System . out . println (i);  for (int j = 0; j < i; j++) {  System .out . println (j);  }  }  System .out . println (" Goodbye !");  } | 0 | 0 | 0 |
| 2 | 1 | n-0+1 | n+1‎ |
| 3 | 1 | n | n |
| 4 | 1 |  |  |
| 5 | 1 |  |  |
| 6 | 0 | 0 | 0 |
| 7 | 0 | 0 | 0 |
| 8 | 1 | 1 | 1 |
| 9 | 0 | 0 | 0 |
|  | Total operation | n2+4n+2 | | |
|  | Big oh | O(n2) | | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | statements | S/E | freq | total |
| 1 | void func4 (int n) {  int m = 1;  while ( m <= n ) {  system . out . println (m);  i = n;  while (i > 0 ) {  system .out . println (i);  i = i / 2;  }  m++;  }  } | 0 | 0 | 0 |
| 2 | 1 | 1 | 1 |
| 3 | 1 | n-1+1+1 | n+1 |
| 4 | 1 | n |  |
| 5 | 1 |  |  |
| 6 | 0 | n(logn+2) | nlogn+2n |
| 7 | 0 | n(logn+1)‎ | nlogn+n |
| 8 | 1 | n(logn+1)‎ | nlogn+n |
| 9 | 0 | 0 | 0 |
| 10 |  | n | n |
| 11 |  | 0 | 0 |
| 12 |  | 0 | 0 |
|  | Total operation | 3nlogn+8n+2 | | |
|  | Big oh | O(nlogn) | | |

# Problem3:

## 1-

**public** **class** Test {

**public** **static** **void** main (String [] args){

**long** selection,bubble,quick, time1,time2;

**for** (**int** i=10000; i<= 50000; i=i+10000){

**double**[] selectionSort= **new** **double**[i];

**double**[] bubbleSort = **new** **double** [i];

**double**[] quickSort = **new** **double** [i];

selection=0;

bubble=0;

quick=0;

**for**(**int** j=0; j<i; j++){

selectionSort[j]=bubbleSort[j]=quickSort[j]=Math.*random*();;

}

**for** (**int** x=0 ; x<100 ; x++){

time1=System.*nanoTime*();

Sort.*selectionSort*(selectionSort,i);

time2= System.*nanoTime*();

selection = selection+(time2-time1);

time1=System.*nanoTime*();

Sort.*bubbleSort*(bubbleSort,i);

time2= System.*nanoTime*();

bubble = bubble+(time2-time1);

time1=System.*nanoTime*();

Sort.*quickSort*(quickSort,i);

time2= System.*nanoTime*();

quick = quick +(time2-time1);

}

System.***out***.println("Averge of selection with size:" + i+ " is: " + ((**double**)selection/100)/1000000);

System.***out***.println("Averge of bubbleSort with size:" + i+ " is: " + ((**double**)bubble/100)/1000000);

System.***out***.println("Averge of quick with size:" + i+ " is: " + ((**double**)quick/100)/1000000);

}

}

}

## 2-

|  |  |  |  |
| --- | --- | --- | --- |
| Size\sort type | Selection sort | Bubble sort | Quick sort |
| 10000 | 38.5 | 57.79 | 0.11 |
| 20000 | 220.46 | 324.78 | 0.09 |
| 30000 | 359.51 | 575.7 | 0.12 |
| 40000 | 625.48 | 951.5 | 0.9 |
| 50000 | 956.38 | 1464.28 | 0.13 |

## 3-

Quicksort 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 oh (n2) .