**Problem 1:**

5n2+2n+1⩽8n2

n0=1

O(n2) c=8

*n*2+*nlog*(*n*) -> *O*(*n*2)

Prove:

*n*2+*nlog*(*n*)⩽*n*2+*n*2

⩽2n2

*O*(*n*2) *c*=2 *n*0=1

2*n*3∉*O*(*n*2)

*f*(*n*)=*O*(*n*3)

*g*(*n*)=*O*(*n*2)

*f*(*n*)≠*g*(*n*)

**a)** 0.1n2

**b)** *n*2

True.

False, because 100 *n*3+8*n*2+5 , *n O*(*n*3)

log*a*(*n*) ∈ *O*(log*b*(*n*)) *a,b*>0 is false.

Ex. let a = 3 , b = 5 , n = 1000

Log3 1000 ≈ 6

Log5 1000 ≈ 4

*an* ∉ *O*(*bn*) *a*>*b*>0 true,

Ex. let a = 3 , b = 2 , n =10

310=1000

210=100

**Problem 2:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| total | ferq | s/e | Statement | # |
| 1 | 1 | 1 | int sum = 0 | 1 |
| n+1 | n+1 | 1 | for( int i = 1 ; i <= n ; i++ ) | 2 |
| ((n+1)2)/4)-1 | ((n+1)2)/4)-1 | 1 | for( int j = 0 ; j < 2 \* i ; j++ ) | 3 |
| (4(n+1)2)/4)-1 | (4(n+1)2)/4)-1 | 1 | sum += j | 4 |
| 1 | 1 | 1 | return sum; | 5 |
| (3n2+10n+1)/4  O(n2) | total | | | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| total | ferq | s/e | Statement | # |
| n3+1 | n3+1 | 1 | for( int i = 0; i < n \* n \* n ; i++ ) | 1 |
| n3 | n3 | 1 | System.out.println(i); | 2 |
| n4-n3 | n3(n-2-1) | 1 | for( int j = 2 ; j < n ; j++ ) { | 3 |
| n4-n3-1 | n4-2n3+n3-1 | 1 | System.out.println(j); } | 4 |
| 1 | 1 | 1 | System.out.println("End"); | 5 |
| 2n4+1  O(n2) | total | | | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| total | ferq | s/e | Statement | # |
| 1 | 1 | 1 | int k = 100, sum = 0; | 1 |
| n+1 | n+1 | 1 | for(int i = 0; i < n ; i++) | 2 |
| 100 | 100n | 1 | for(int j = 1 ; j <= k ; j++) { | 3 |
| 99n | 99n | 1 | sum = i + j; | 4 |
| 99n | 99n | 1 | System.out.println(sum); | 5 |
| 0 | 0 | 0 | } | 6 |
| 299n+2 O(n) | total | | | |

**Problem 3:**

1. O(nlog(n))
2. The best-case all number odd O(nlog(n)). The worst-case all number even O(n2).

a- *O*(*n*)

b- *O*(*n*)

c- *O*(*n*2)

d- *O*(*n*2)

e- *O*(*n*3)

**Problem 5:**

O(n2)

**Problem 6:**

import java.util.\*;

public class Main{

public static void fill(double[] A , int n){

for (int i = 0; i < n; i++){

A[i] = Math.random();

}

}

public static double expSelectionSort (double[] A , int n){

double total = 0;

for (int i = 0; i < 100; i++){

double[] b = Arrays.copyOf(A, A.length);

long befor = System.nanoTime();

Sort.selectionSort(b, n);

long after = System.nanoTime();

total = total + (after - befor);

}

total /= 1000000;

return total / 100;

}

public static double expBubbleSort (double[] A , int n){

double total = 0;

for (int i = 0; i < 100; i++){

double[] b = Arrays.copyOf(A,A.length);

long befor = System.nanoTime();

Sort.bubbleSort(b, n);;

long after = System.nanoTime();

total = total + (after - befor);

}

total /= 1000000;

return total / 100;

}

public static double expQuickSort (double[] A , int n){

double total = 0;

for (int i = 0; i < 100; i++){

double[] b = Arrays.copyOf(A, A.length);

long befor = System.nanoTime();

Sort.quickSort(b, n);;

long after = System.nanoTime();

total = total + (after - befor);

}

total /= 1000000;

return total / 100;

}

public static void main (String[] args){

int[] data = new int[5];

double A[] = new double[500000];

fill(A, 500000);

System.out.println("Data \t\t SelectionTime\t\t BubbleTime \t\t Quick Time ");

for (int i = 0 ; i < 5 ; i++){

data[i] = (i+1)\*10000;

System.out.println(data[i] + " \t\t " + expSelectionSort(A, data[i]) + " \t\t "

+expBubbleSort(A, data[i]) + " \t\t " + expQuickSort(A,data[i]));

}

}

}

**Table**

|  |  |  |  |
| --- | --- | --- | --- |
| **Quick Sort** | **Bubble Sort** | **Selection Sort** | **Data** |
| 0.88 | 126.86 | 40.39 | 10000 |
| 1.38 | 543.86 | 157.64 | 20000 |
| 2 | 1272.34 | 363.28 | 30000 |
| 2.58 | 2295.32 | 636.68 | 40000 |
| 3.34 | 3630.29 | 979.74 | 50000 |

**Graph**

1. The Quick sort is the fastest one.
2. Selection sort is faster than Bubble sort