Design & Analysis of Algorithm

Here I am about to give my empirical analysis of sorting algorithms namely bubble sort, selection sort and insertion sort respectively.

So few general things to be noted , since there is process of getting input and putting input across every program I have created separate class for getting input as well as printing input.

So here is the code for my class named getinput – where I have accomplished the task of printing input as well as getting inputs from user. Here I have created four methods- namely

1. Worstcaseinput –(takes an array assigns first number=1000(for n=1000) and last i.e. 1000th element = 1)
2. Bestcaseinput–(takes an array assigns first number=1(for n=1000) and last i.e. 1000th element = 1000)
3. Getrandominput-(takes an array of size n and appends random numbers to array)
4. Doprintinput(prints our sorted array)

Code:

package algorithmsDAA;

import java.util.Random;

import java.util.Scanner;

public class getinput{

Scanner s2 = new Scanner(System.*in*);

int[] worstcaseinput(int n){

/\* this function will scan the number from user , i.e n . and for value of n it will declare the size of array, and it will set first element =n ,

\* second element = n-1 , so on upto last element =1\*/

int i,j;

int []a = new int[n];

for(i=0 , j=n;i<n && j>=1 ;i++ ,j--){

a[i]=j;

}

return a;

}

int[] besttcaseinput(int n){

/\* this function will scan the number from user , i.e n . and for value of n it will declare the size of array, and it will set first element =1 ,

\* second element = 2 , so on upto last element =n\*/

int i,j;

int []a = new int[n];

for(i=0 , j=1;i<n && j<=n ;i++ ,j++){

a[i]=j;

}

return a;

}

int[] getrandominput(int n){

/\* this will scan n random variables \*/

Random r1 = new Random();

int i;

int []a = new int[n];

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

a[i]=r1.nextInt(n);

}return a;}

void doprintinput(int b[], int c){

int j;

for(j=0;j<c;j++){

System.*out*.print(b[j] + " , ");

}

System.*out*.print("completed Sorting!!");

System.*out*.println();

}

}

Now on the ground of this let’s start the analysis of algorithm.

Theoretically, analysis of above stated algorithms are:

| Algorithm | | Time Complexity | | | Space Complexity |
| --- | --- | --- | --- | --- | --- |
|  | Best | | Average | Worst | Worst |
| [Quicksort](http://en.wikipedia.org/wiki/Quicksort) | O(n log(n)) | | O(n log(n)) | O(n^2) | O(log(n)) |
| [Mergesort](http://en.wikipedia.org/wiki/Merge_sort) | O(n log(n)) | | O(n log(n)) | O(n log(n)) | O(n) |
| [Timsort](http://en.wikipedia.org/wiki/Timsort) | O(n) | | O(n log(n)) | O(n log(n)) | O(n) |
| [Heapsort](http://en.wikipedia.org/wiki/Heapsort) | O(n log(n)) | | O(n log(n)) | O(n log(n)) | O(1) |
| [Bubble Sort](http://en.wikipedia.org/wiki/Bubble_sort) | O(n) | | O(n^2) | O(n^2) | O(1) |
| [Insertion Sort](http://en.wikipedia.org/wiki/Insertion_sort) | O(n) | | O(n^2) | O(n^2) | O(1) |
| [Selection Sort](http://en.wikipedia.org/wiki/Selection_sort) | O(n^2) | | O(n^2) | O(n^2) | O(1) |
| [Shell Sort](http://en.wikipedia.org/wiki/Shellsort) | O(n) | | O((nlog(n))^2) | O((nlog(n))^2) | O(1) |
| [Bucket Sort](http://en.wikipedia.org/wiki/Bucket_sort) | O(n+k) | | O(n+k) | O(n^2) | O(n) |
| [Radix Sort](http://en.wikipedia.org/wiki/Radix_sort) | O(nk) | | O(nk) | O(nk) | O(n+k) |

Below Table Source: <http://bigocheatsheet.com/>

ExcellentGoodFairBad |Horrible

# Bubblesort – algorithm:

Code:

package algorithmsDAA;

import java.util.Scanner;

class bubblesorting{

int outer,inner,temp;

int[] bubblesorts(int[] a,int n){

for(outer=n-1;outer>0;outer--){

for(inner=0;inner<outer;inner++){

if(a[inner]>a[inner+1]){

temp = a[inner];

a[inner]= a[inner+1];

a[inner+1]=temp;

}

}

}

return a;

}

}

public class bubblesort {

private static Scanner *s1*;

public static void main(String args[]){

long lStartTime = System.*nanoTime*();

*s1* = new Scanner(System.*in*);

int n;

System.*out*.println("Enter the number of numbers you wish to print.");

n=*s1*.nextInt();

int []a = new int[n];

getinput ssrt = new getinput();

a = ssrt.getrandominput(n);

//a=ssrt.besttcaseinput(n);

//a=ssrt.worstcaseinput(n);

System.*out*.println("The last element is " + a[n-1]);

bubblesorting b1 = new bubblesorting();

a = b1.bubblesorts(a, n);

ssrt.doprintinput(a, n);

long lEndTime = System.*nanoTime*();

long difference = lEndTime - lStartTime;

System.*out*.println("Elapsed time for program is " + difference + " nano seconds or say "+ (difference/1000000000) + " seconds ");

}

}

## Code analysis: Bubble Sort

Worst case scenario: user entering the reverse number than sorted one, e.g: 10,9,8,7,6,5,4,3,2,1 for user’s input = 10.

|  |  |  |
| --- | --- | --- |
| Input values | Time taken in nanoseconds | Time taken in Seconds |
| 10 to 1 | 712614967 | 0.7 |
| 100 to 1 | 1068117273 | 1.06 |
| 1000 to 1 | 1112109037 | 1.11 |
| 10000 to 1 | 1850844495 | 1.85 |
| 1,00,000 to 1 | 24560351649 | 24 |
| 2,00,000 to 1 | 95530448900 | 95 |
| 5,00,000 to 1 | 591683009364 | 591 |
| 10,00,000 to 1 | 4062385508665 | 4062(67.7 minutes) |

It takes huge amount of time if I go for1, 00, 00,000(which is the maximum limit for int variable in java) element array using bubble sort & it will practically gobble up all your patience to wait for such time, I waited for more than 12 hours in my laptop using Pentium processor but seriously would conduct it some day and bring it on to you!

Best case scenario: user entering the reverse number than sorted one, e.g: 1,2,3,4,5,6,7,8,9,10 for user’s input = 10.

|  |  |  |
| --- | --- | --- |
| Input values | Time taken in nanoseconds | Time taken in Seconds |
| 1 to 10 | 609173500 | 0.6 |
| 1 to 100 | 973117095 | 0.97 |
| 1 to 1000 | 1331609920 | 1.3 |
| 1 to 10000 | 1426020694 | 1.42 |
| 1 to 1,00,000 | 22120863599 | 22 |
| 1 to 2,00,000 | 84834182822 | 84 |
| 1 to 5,00,000 | 591683009364 | 591 |

Random case scenario:

|  |  |  |
| --- | --- | --- |
| Input values | Time taken in nanoseconds | Time taken in Seconds |
| 10 | 677723296 | 0.67 |
| 100 | 1098430552 | 1.09 |
| 1000 | 1971144593 | 1.97 |
| 10000 | 1788450213 | 1.78 |
| 1,00,000 | 40505548760 | 40 |
| 2,00,000 | 166551028697 | 166 |
| 5,00,000 | 5167683009364 | 516 |

# Selection sort – Algorithm:

Code:

package algorithmsDAA;

import java.util.Scanner;

class selectionsorting{

int outer,inner,min;

int[] selectionsorts(int[] a,int n){

for(outer=0;outer<n-1;outer++){

min=outer;

for(inner=outer+1;inner<n;inner++)

{

if(a[inner]<a[min]){

min=inner;

}

}

int temp = a[outer];

a[outer] = a[min];

a[min]=temp;

}

return a;

}

}

public class selectionsort {

private static Scanner *s1*;

public static void main(String[] args) {

// TODO Auto-generated method stub

*s1* = new Scanner(System.*in*);

long lStartTime = System.*currentTimeMillis*();

getinput ssrt = new getinput();

int n;

System.*out*.println("Enter the number of numbers you wish to print.");

n=*s1*.nextInt();

int []a = new int[n];

a = ssrt.getrandominput(n);

//a=ssrt.besttcaseinput(n);

//a=ssrt.worstcaseinput(n);

selectionsorting sr = new selectionsorting();

a = sr.selectionsorts(a, n);

ssrt.doprintinput(a, n);

long lEndTime = System.*currentTimeMillis*();

long difference = lEndTime - lStartTime;

System.*out*.println("Elapsed time for program is " + difference + " or say "+ (difference/1000) + " seconds ");

}

}

## Code analysis: Selection Sort

Worst case scenario: user entering the reverse number than sorted one, e.g: 10,9,8,7,6,5,4,3,2,1 for user’s input = 10.

|  |  |  |
| --- | --- | --- |
| Input values | Time taken in nanoseconds | Time taken in Seconds |
| 10 to 1 | 712643313 | 0.7 |
| 100 to 1 | 1022001502 | 1 |
| 1000 to 1 | 2103993240 | 2 |
| 10000 to 1 | 2834146656 | 2 |
| 1,00,000 to 1 | 21978790724 | 21 |
| 2,00,000 to 1 | 88889375715 | 88 |
| 5,00,000 to 1 | 563597143978 | 563 |

Best case scenario: user entering the reverse number than sorted one, e.g: 1,2,3,4,5,6,7,8,9,10 for user’s input = 10.

|  |  |  |
| --- | --- | --- |
| Input values | Time taken in nanoseconds | Time taken in Seconds |
| 1 to 10 | 748254390 | 0.7 |
| 1 to 100 | 1001057605 | 1.01 |
| 1 to 1000 | 2420225371 | 2 |
| 1 to 10000 | 2465347561 | 2 |
| 1 to 1,00,000 | 22049701559 | 22 |
| 1 to 2,00,000 | 88174586403 | 88 |
| 1 to 5,00,000 | 513683008961 | 516 |

Random case scenario:

|  |  |  |
| --- | --- | --- |
| Input values | Time taken in nanoseconds | Time taken in Seconds |
| 10 | 763407365 | 0.7 |
| 100 | 1198266682 | 1 |
| 1000 | 1542462782 | 1.5 |
| 10000 | 1764031469 | 1.7 |
| 1,00,000 | 24378228101 | 24 |
| 2,00,000 | 91154929811 | 91 |
| 5,00,000 | 591683009364 | 591 |

## Insertion Sort Algorithm:

package algorithmsDAA;

import java.util.Scanner;

class insertionsorting{

int[] insertionSorts(int arr[], int n)

{

int i, key, j;

for (i = 1; i < n; i++)

{

key = arr[i];

j = i-1;

while (j >= 0 && arr[j] > key)

{

arr[j+1] = arr[j];

j = j-1;

}

arr[j+1] = key;

}

return arr;

}

}

public class insertionsort {

private static Scanner *s1*;

public static void main(String[] args) {

// TODO Auto-generated method stub

long lStartTime = System.*currentTimeMillis*();

*s1* = new Scanner(System.*in*);

getinput ssrt = new getinput();

int n;

System.*out*.println("Enter the number of numbers you wish to print");

n=*s1*.nextInt();

int []a = new int[n];

a = ssrt.getrandominput(n);

//a=ssrt.besttcaseinput(n);

//a=ssrt.worstcaseinput(n);

insertionsorting a1 = new insertionsorting();

a = a1.insertionSorts(a,n);

ssrt.doprintinput(a, n);

long lEndTime = System.*currentTimeMillis*();

long difference = lEndTime - lStartTime;

System.*out*.println("Elapsed time for program is " + difference + " milliseconds or say "+ (difference/1000) + " seconds ");

}

}

# Code analysis: Insertion Sort

Worst case scenario: user entering the reverse number than sorted one, e.g: 10,9,8,7,6,5,4,3,2,1 for user’s input = 10.

|  |  |  |
| --- | --- | --- |
| Input values | Time taken in nanoseconds | Time taken in Seconds |
| 10 to 1 | 781170628 | 1 |
| 100 to 1 | 1065121378 | 1 |
| 1000 to 1 | 1503633439 | 1.5 |
| 10000 to 1 | 1988202135 | 1.98 |
| 1,00,000 to 1 | 17793814478 | 17 |
| 2,00,000 to 1 | 69104567005 | 69 |
| 5,00,000 to 1 | 408123649343 | 408 |

Best case scenario: user entering the reverse number than sorted one, e.g: 1,2,3,4,5,6,7,8,9,10 for user’s input = 10.

|  |  |  |
| --- | --- | --- |
| Input values | Time taken in nanoseconds | Time taken in Seconds |
| 1 to 10 | 795865664 | 1 |
| 1 to 100 | 929077924 | 1 |
| 1 to 1000 | 1519367021 | 2 |
| 1 to 10000 | 2473097789 | 3 |
| 1 to 1,00,000 | 3981071612 | 3 |
| 1 to 2,00,000 | 14163403845 | 14 |
| 1 to 5,00,000 | 83860455796 | 83 |

Random case scenario:

|  |  |  |
| --- | --- | --- |
| Input values | Time taken in nanoseconds | Time taken in Seconds |
| 10 | 741468419 | 0.7 |
| 100 | 953781108 | 0.9 |
| 1000 | 1488347043 | 1.48 |
| 10000 | 1635404444 | 1.6 |
| 1,00,000 | 10729396868 | 10 |
| 2,00,000 | 39919626926 | 39 |
| 5,00,000 | 245319515887 | 245 |

To compute sorting of 10, 00,000 elements in worst case that is starting from 10, 00,000 and going up to 1, it takes infinite number of time to compute in bubble sort, insertion sort or selection sort, so I took help of heap sort which is said to be best sort and all the sorting process were completed in 332661 milliseconds or say 332 seconds.

## Conclusion

Thus after observing above data one thing can be concluded that, for small inputs like 10,100,1000,10,000 the time we get can be varying on the basis of time we take to input the number thus, here I am drawing a table for time taken by each algorithm to sort 5 lakh elements in worst case, best case and random case, as well as the same figures from another pc having better processor.

Sys-Pentium(1) = Intel Pentium(R) CPU B950 @2.10Ghz ,64 bit OS x64bit processor, 6gb ram.

Sys-i3(2) = Intel (R)Core(Tm) i3-3210CPU 3.20Ghz , 32 bit OS x64bit processor, 4gb ram.

Computing array of 2, 00,000 elements:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Algorithms | Bubble Sort | | | Selection Sort | | | Insertion Sort | | |
| Cases | B.C [O(n)] | R.C  [O(n^2)] | W.C  [O(n^2)] | B.C  [O(n^2)] | R.C  [O(n^2)] | W.C  [O(n^2)] | B.C  [O(n)] | R.C  [O(n^2)] | W.C  [O(n^2)] |
| Sys-Pentium | 84 | 166 | 95 | 88 | 91 | 88 | 14 | 39 | 69 |
| Sys- i3 | 47 | 96 | 58 | 53 | 52 | 54 | 8 | 24 | 41 |

Computing array of 5, 00,000 elements:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Algorithms | Bubble Sort | | | Selection Sort | | | Insertion Sort | | |
| Cases | B.C [O(n)] | R.C  [O(n^2)] | W.C  [O(n^2)] | B.C  [O(n^2)] | R.C  [O(n^2)] | W.C  [O(n^2)] | B.C  [O(n)] | R.C  [O(n^2)] | W.C  [O(n^2)] |
| Sys-Pentium | 516 | 1057 | 591 | 513 | 526 | 563 | 83 | 245 | 408 |
| Sys- i3 | 294 | 603 | 364 | 327 | 330 | 331 | 50 | 155 | 256 |

P.S time is in **seconds**. And all the values displayed here are the time taken to compute **5, 00,000** elements in an array.

Comparing 10, 00,000 elements, worst cases for all the three algorithms:

|  |  |  |  |
| --- | --- | --- | --- |
| Algorithm | Bubble sort | Selection sort | Insertion sort |
| Sys-Pentium | 2382(39.7 minutes) | 2071(34.51 minutes) | 1662(27.7 minutes) |
| Sys-i3 | 1453(24.2minutes!) | 1342(22.36 minutes) | 1027(17.11 minutes) |

Contribute =

<https://github.com/AbhishekRaval/algorithmsDAA/tree/master/src/algorithmsDAA>