

DESIGN AND ANALYSIS OF ALGORITHMS

EXPERIMENT 1B

ANUSHKA ACHARYA

CSE DS D1 BATCH

UID: 2021700001

AIM: TO FIND RUNNING TIME OF AN ALGORITHM

THEORY:

The goal of the analysis of algorithms is to compare algorithms (or solutions) mainly in terms of running time but also in terms of other factors (e.g., memory, developer effort, etc.) The running time of an algorithm on a particular input is the number of primitive operations or “steps” executed.

Insertion sort is a simple sorting algorithm that works similar to the way you sort playing cards in your hands. The array is virtually split into a sorted and an unsorted part. Values from the unsorted part are picked and placed at the correct position in the sorted part.

Characteristics of Insertion Sort:

- This algorithm is one of the simplest algorithm with simple implementation
- Basically, Insertion sort is efficient for small data values
- Insertion sort is adaptive in nature, i.e. it is appropriate for data sets which are already partially sorted.

[Insertion Sort - GeeksforGeeks](#)

Selection sort is a simple and efficient sorting algorithm that works by repeatedly selecting the smallest (or largest) element from the unsorted portion of the list and moving it to the sorted portion of the list. The algorithm repeatedly selects the smallest (or largest) element from the unsorted portion of the list and swaps it with the first element of the unsorted portion. This process is repeated for the remaining unsorted portion of the list until the entire list is sorted. One variation of selection sort is called “Bidirectional selection sort” that goes through the list of elements by alternating between the smallest and largest element, this way the algorithm can be faster in some cases.

The algorithm maintains two subarrays in a given array.

- The subarray which already sorted.
- The remaining subarray was unsorted.

[Selection Sort Algorithm - GeeksforGeeks](#)

ALGORITHMS:

(A) FOR GENERATING RANDOM NUMBERS:

STEP_1: Start. Include the standard c library in the program.

STEP_2: Define an integer x

STEP_3: Run a for loop from 0 to 100000 to generate 100000 numbers

STEP_4: Access the random function rand() inside the loop to generate 100000 random numbers.

STEP_5: Print the numbers and save the text file. Stop

(B) INSERTION SORT:

- STEP_1: Start. Initialise size of the array as 100 to consider first block of 100 elements.
- STEP_2: Open the file where the random numbers are stored to read the numbers.
- STEP_3: Start a while loop for size \leq 100000 so the program executes for all the blocks. In the while loop:
- (1) Initialise time taken as 0. Declare begin time using clock() function.
 - (2) Declare an array of the size given and start reading the random numbers from the file using a for loop.
 - (3) Consider first two elements, compare them. If second element is greater than the first, swap them and put the first element in the sorted subarray.
 - (4) Consider second and third element, compare them and swap if needed. Now if the third element is less than both second and first elements, put it as the first element of sorted subarray by swapping.
 - (5) Repeat steps 3 and 4 until the last element is reached.
 - (6) Print the sorted subarray, increment the size and bring the file pointer back to beginning of the file.
 - (7) Declare end time using clock() function and calculate runtime using begin and end times.
- STEP_4: Stop

(C) SELECTION SORT:

- STEP_1: Start. Declare a function which swaps two elements using a temporary variable.
- STEP_2: Declare a function to perform the selection sort process. In this function:
- (1) Use two for loops to find the minimum element of the array by comparing two consecutive elements.
 - (2) Once you find the minimum element, bring it to the front by swapping it with the first element of the array.
 - (3) Return the sorted array to the main function.
- STEP_5: In the main function, initialise size of array as 100. Initialise time taken as 0. Declare begin time using clock() function.
- STEP_6: Open the file where the random numbers are stored to read the numbers.
- STEP_7: Start a while loop for size \leq 100000 so the program executes for all the blocks. In the while loop:
- (1) Declare an array of the size given and start reading the random numbers from the file using a for loop.
 - (2) Call the sort function to get the sorted array
 - (3) Print the array using for loop, increment size by 100 and bring the file pointer back to beginning of the file.
 - (4) Declare end time using clock() function and calculate runtime using begin and end times.
- STEP_8: Stop

CODE:

```
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
```

```

int main(){
    int x;
    for(int i=0;i<=100000;i++){
        x=rand()%100000;
        printf("%d ",x);
    }
    return 0;
}

```

INSERTION SORT:

```

#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include<time.h>

```

```

void main(){
int pos,swap,size=100;
FILE *fp;
fp = fopen("anu.txt", "r");
while(size<=100000){
double time_taken=0.0;
clock_t begin=clock();
int a[size];
for(int j=0;j<size;j++){
fscanf(fp,"%d",&a[j]);
}
for(int k = 0; k < (size) - 1; k++){
pos=k;
for(int p = k + 1; p < size; p++)
{
if(a[pos] > a[p])
pos=p;
}
if(pos != k)
{
int temp=a[k];
a[k]=a[pos];
a[pos]=temp;
}
}
int i1, element, j1;
for (i1 = 1; i1 < size; i1++) {
    element = a[i1];
    j1 = i1 - 1;
    while (j1>= 0 && a[j1] > element) {
        a[j1 + 1] = a[j1];
        j1 = j1 - 1;
    }
    a[j1 + 1] = element;
}
for(int test1=1;test1<size;test1++){
printf("%d\t",a[test1]);
}
printf("\n\n");
size=size+100;
}

```

```

        fseek(fp,0,SEEK_SET);
clock_t end=clock();
        time_taken = (double)(end-begin)/CLOCKS_PER_SEC;
        printf("Runtime: %f\n",time_taken);
    }
}

```

SELECTION SORT:

```

#include<stdio.h>
#include<stdlib.h>
#include<time.h>

```

```

void swap(int*a , int*b){
    int temp = *a;
    *a = *b;
    *b = temp;
}

```

```

void selSort(int* arr , int size){
    for(int i=0;i<size-1;i++){
        int minId = i;
        for(int j=i+1;j<size;j++){
            if(arr[j]<arr[minId]){
                minId = j;
            }
        }
        if(i!=minId){
            swap(&arr[i],&arr[minId]);
        }
    }
}

```

```

int main(){
int size=100;
FILE *fp;
fp = fopen("anu.txt","r");
double time_taken=0.0;
clock_t begin=clock();
while(size<=100000){
int a[size];
for(int j=0;j<size;j++){
fscanf(fp,"%d",&a[j]);
}
selSort(a,size);
for(int test1=1;test1<size;test1++){
printf("%d\t",a[test1]);
}
printf("\n\n");
size=size+100;
fseek(fp,0,SEEK_SET);
clock_t end=clock();
        time_taken = (double)(end-begin)/CLOCKS_PER_SEC;
        printf("Runtime: %f\n",time_taken);
    }
}

```

OUTPUT:

Insertion sort output for first three blocks:

```
C:\Users\User\Desktop\DAO\gcc Insert.c -o Insert

C:\Users\User\Desktop\DAO\gcc Insert
3624 4364 4852 4938 5122 5326 5693 7644 9667 19188 18207 12945 15493 15561 16876
16193 17541 20105 22885 23139 23610 25408 27730 28787 30102 32647 32766 36553
36560 36686 38581 38578 38705 41524 41925 42028 42790 43433 44508 44536 46502 46839
49658 50209 52352 53755 54023 54698 55569 57230 57426 57843 58323 59995 61807 62062
64614 67560 67570 69034 70464 71995 72415 74771 75058 75832 77542 77677 78716 78734
79222 80601 80891 85761 86067 86138 87153 88224 88421 88658 90808 90814 90827 90961
91413 91687 91844 91968 92297 92501 92615 93293 95323 96345 97492 97828 98531 99403

Runtime: 0.015000
1207 2361 3461 3624 3856 3939 4364 4454 4567 4852 4938 5122 5326 5397 5442
12945 6902 7211 7215 7278 7644 8726 9667 9973 10188 10287 11669 11881 12554
15805 14053 15492 15501 16076 16293 17561 20105 20275 21848 22285 22783 23139 23610
25163 25408 25960 27730 28787 29049 29096 30102 31555 32049 32647
32694 32766 34830 36553 36560 36686 38884 38581 38578 38765 39116 40787 41075 41232
41524 41925 42028 42448 42790 43433 43945 44508 44536 45014 46184 46802 46583 46839
48536 49639 49658 50209 50760 50867 52352 52525 53609 53727 53755 54023 54698 54704
55500 55569 55614 56381 57230 57426 57572 57843 57847 58034 58070 58323 58542 58692
58202 58995 60511 60791 61007 62062 62304 62646 64614 67560 67570 69034 69100 70147
70331 70343 70464 71389 71995 72415 72554 73422 74434 74771 74846 75058 75994 75832
76491 77263 77542 77644 77677 77823 77990 78716 78734 79222 80024 80536 80601 80801
81082 81740 82149 82527 84361 84754 85002 85430 85671 85764 86087 86103 86138 86409
86729 87153 88123 88224 88421 88658 90808 90814 90827 90961 91413 91687 91844 91896
91908 92297 92501 92615 93293 94822 95323 96345 96944 97492 97828 97916 98505 98531
99145 99414

Runtime: 0.031000
896 987 989 1072 1207 2361 3243 3461 3624 3856 3871 3896 3930 3939 4364
4454 4567 4852 4938 5122 5326 5397 5442 5785 5971 6259 6902 7211 7215
7278 7644 8726 8958 9000 9841 9667 9838 9973 10068 10188 10967 11047 11368
11497 11669 11881 12554 12945 13112 13128 13357 14453 14218 14817 15492 15501 16076
16293 16393 17227 17561 17901 19311 20105 20275 21848 22007 22100 22285 22572 22783
23139 23610 24067 25163 25163 25764 25960 25960 26602 27205 27730 28787 28961 29190
29336 29355 29600 29951 30049 30096 30102 31111 31469 31556 32405 32647 32694 32766
33783 34607 34830 35078 35138 35833 35836 36246 36553 36560 36606 36884 37366 38353
38501 38578 38765 39116 40194 40787 41075 41232 41297 41524 41925 42828 42448 42479
42790 43433 43945 45008 44536 45014 46184 46384 46502 46583 46614 46839 47101 47345
48536 49639 49658 50209 50760 50867 52352 52525 53609 53727 53755 54023 54698 54704
53027 54023 54277 46098 54704 55500 55569 55596 55614 56381 57230 57426 57572 57843
57843 57847 58034 58070 58323 58542 59282 59954 59995 60511 60791 60822 60791 61807
```

Selection sort output for first three blocks:

```

C:\Users\user\Desktop\DA>gcc select.c -o select

Runtime: 0.070800

C:\Users\user\Desktop\DA>select

3624
4364 4852 4938 5122 5326 6002 7644 9667 10188 10207 12945 15492 15501 16076 16293 17561 20105 22285 23139 23610
25940 25968 27200 28782 30102 32647 32766 36553 36660 36606 38501 38578 38765 41524 41925 42028 42790 43433 44508 44536
46502 46839 49658 50209 52352 53755 54023 54698 55569 57230 57426 57843 58323 59965 61807 62862 64614 67660 67970 69394
70464 71995 72415 74771 75058 78328 77542 77777 78716 78734 79222 80601 80891 85961 86067 86138 87153 88224 88421 88658
90808 90814 90827 90961 91413 91687 91844 91908 92297 92501 92616 93293 95323 96345 97492 97828 98531 99414

Runtime: 0.000000

1207
2361 3461 3624 3856 3939 4364 4454 4567 4825 4938 5122 5326 5397 5442 5795 6902 7211 7215 7278 7644
7261 9667 9973 10188 10207 11669 11881 12554 12945 14053 15492 15501 16076 16293 17561 20105 20275 21848 22285 22783
23139 23610 25163 25940 25968 27200 28782 28961 29336 29535 30049 30096 30102 31556 32495 32647 32694 32766 34830 36553
36319 36606 36884 38581 38578 38765 39116 40787 41075 41232 41925 42028 42448 42790 43433 43945 44508 44536 45416
46184 46502 46583 46839 48536 49658 50209 50768 50867 52352 52525 53609 53727 53755 54023 54698 54704 55500 55569
56414 56931 57426 57426 57872 57843 57847 58034 58074 58323 58542 59612 59902 59955 60111 60791 61807 62062 62304 62646
64614 67500 67570 69034 69108 70417 70311 70343 70664 71389 71995 72415 72554 73422 74434 74771 74846 75058 75804 75832
76491 77263 77542 77644 77677 77823 77990 78116 78734 79222 80024 80536 80601 80891 81082 81419 82149 82527 84361 84754
85002 85430 85671 85764 86007 86103 86138 86409 86729 87123 88123 88224 88421 88588 88608 90814 90827 90961 91413 91687
91044 91096 91900 92297 92501 92616 93293 94022 95323 96345 96944 97492 97020 97916 99050 99231 99145 99414

Runtime: 0.000310

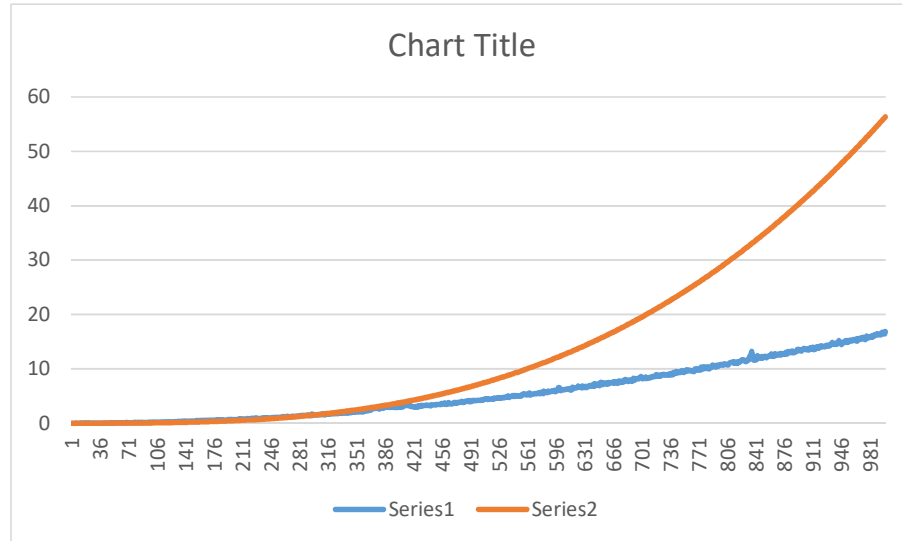
896
907 989 1072 1207 2361 3243 3461 3624 3758 3876 3871 3896 3930 3939 4044 4454 4567 4687 4953 4938 5122 5326
5397 5442 5705 5971 6259 6902 7211 7215 7278 7644 8782 8858 9000 9361 9667 9838 9773 10068 10188 10207
10347 11368 11497 11669 11881 12554 12945 13112 13128 13153 14053 14218 14817 15492 15501 16076 16293 16383 17227 17561
17981 19311 20105 20275 21848 22200 22285 22672 22783 23139 23610 24067 25140 25561 25764 25940 25968 26682 27851
27730 28787 28961 29190 29336 29355 29600 29951 30049 30896 30102 31111 31469 31556 32495 32647 32694 32766 33783 34007
35078 35078 35136 35423 35936 36246 36553 36560 36606 36084 37366 38253 38501 38578 38765 39116 40146 40787 41075 41232
41297 41524 41925 42028 42448 42470 42790 43433 43945 44508 44536 44614 46184 46384 46502 46583 46614 46839 47101 47345
48336 49587 49639 49658 50014 50209 50768 50867 50872 51232 52352 52525 53609 53727 53755 53972 54023 54227 54698 54704 55500
55569 55596 55614 56381 57230 57426 57552 57572 57843 57847 58034 58079 58323 58542 58992 59202 59954 59965 60511 60522
60791 61807 61927 62662 62304 62646 63902 64095 64597 64614 67387 67560 67570 67960 69034 69108 70411 70147 70331 70343
70646 70499 70848 71389 71995 72382 72415 72554 73422 73522 74434 74727 74771 74846 75058 75804 75832 76491 77263 77542
77677 77923 77990 78116 78734 79222 79290 79650 80024 80536 80601 80891 81021 81082 81230 81419 81455 82149 82527
83063 84251 84361 84754 85002 85064 85108 85430 85671 85764 85910 86007 86103 86138 86409 86513 86729 87153 87231 88123
88224 88421 88658 89079 90286 90808 90814 90827 90961 91413 91433 91687 91844 91896 91900 92297 92501 92598 92616 93293
93405 94182 94282 95323 95405 95660 96345 96944 97004 97492 97828 97916 98505 98905 99121 99145 99414 99770

Runtime: 0.000780

C:\Users\user\Desktop\DA>

```

GRAPH:



OBSERVATION:

In this graph, the blue line represents selection sort whereas the red curve represents insertion sort algorithm. It can be seen clearly then selection sort has a higher runtime than the insertion sort algorithm. This is because in selection sort the array is scanned repeatedly to find the minimum element and then the minimum element is brought to the front by swapping. So the number of comparison needed is more which gives this algorithm a time complexity as $O(n^2)$ in all cases. Opposed to this, in insertion sort, consecutive elements are compared and swapping occurs as soon as the order is disrupted. This leads to less comparisons and time complexity of $O(n)$ in best case scenario. This is why, insertion sort is more efficient than selection sort algorithm.

CONCLUSION:

From this experiment I learnt how to perform selection and insertion sort algorithms. I used the sorting techniques on 100000 random numbers and found out that runtime for insertion sort is less than select sort which helped me analyse that insertion sort is the algorithm more efficient between the two.