DAA ASSIGNMENT

Merlin Xavier
Roll No: 44

1) Compute the time complexity of following snippet of codes:

a)
```
int i, j, k=0;
for(i=n/2 ; i<=n; i++){             — n
    for(j=2 ; j<n; j=j*2){          — log₂n
        k=k+n/2;
    }
}
```

Time complexity = $\underline{\underline{O(n \log n)}}$

b)
```
int a=0; i=N;
while (i>0){
    a+=i;          } — log₂n
    i/=2;
}
```

Time complexity = $\underline{\underline{O(\log n)}}$

c)
```
void fun (int n) {
    for(int i=0 ; i<n ; i++)                 — n
        for(int j=n; j>0 ; j=j/2)            — log₂n
            printf("*");
}
```

Time complexity = $O(\underline{\underline{n \log n}})$

2) Solve the recurrence relation :

$T(n) = T(n/2) + \log_2 n$ where $T(1) = 1$

Ans: $T(n) = T(n/2) + \log_2 n$

$T(n/2) = T(n/4) + \log(n/2)$

$T(n/4) = T(n/8) + \log(n/4)$

$\vdots$

$T(n) = T\left(\frac{n}{2^k}\right) + \Sigma \log_2\left(\frac{n}{2^i}\right)$

$= \log_2\left(\frac{n}{2^i}\right) \Rightarrow \log n - \log_2 2^i$

$= \log n - i$

$T(1) = T\left(\frac{n}{2^k}\right)$

where, $\frac{n}{2^k} = 1$, $n = 2^k$, $\log n = k$

$\therefore T(n) = T(1) + \Sigma(\log n - i)$

$= T(1) + \Sigma \log n - \Sigma i$

$\Rightarrow 1 + k \log n - \frac{k(k-1)}{2}$

Since $k = \log n$,

$T(n) = 1 + \log n \cdot \log n - \frac{\log n(\log n - 1)}{2}$

$= \underline{O(\log^2 n)}$.

3) Algorithm A performs $10n^2$ basic operations, and algorithm B performs $300 \log n$ basic operations. For what value of n does algorithm B start to show its better performance?

Ans: Algorithm A : $10n^2$

Algorithm B : $300 \log n$

We find when : $300 \log n < 10n^2$

Divide by 10 :

$= 30 \log n < n^2$

n	$A = n^2$	$B = 30 \log n$
2^2	16	60
2^3	64	90
2^4	256	120

$\therefore$ Algorithm B is better for

$n \geq 16$

4) Write programs in any language to implement the following sorting algorithms:

1. Bubble sort
2. Selection sort
3. Insertion sort

Maintain a counter variable to record the number of iterations executed by: the outer loop and the inner loop.

Using this counter, determine and compare the number of iterations required for:

- Best case
- Average case
- Worst case

Ans: 1) <u>Bubble Sort</u>

```
#include <stdio.h>
void bubblesort (int a[], int n){
    int i, j, temp;
    int outer =0, inner =0;
    for(i=0 ; i<n-1 ; i++){
      outer ++;
      for(j=0; j<n-i-1; j++){
        inner++;
```

```
          if(a[j]>a[j+1]) {
              temp= a[j];
              a[j]=a[j+1];
              a[j+1]= temp;
          }
      }
  }
  printf ("Outer loop count = %d \n", outer);
  printf ("Inner loop count = %d \n", inner);
}
```

2) <u>Selection sort</u>

```
#include <stdio.h>
void selectionsort (int a[], int n){
    int i , j, min , temp;
    int outer =0, inner= 0;
    for(i=0;i<n-1 ; i++) {
        outer ++;
        min= i ;
        for(j=i+1 ;j<n ;j++) {
            inner++;
            if(a[j]<a[min]) {
                min= j;
            }
        }
        temp = a[i];
        a[i]= a[min];
        a[min]= temp;
      }
    printf ("Outer loop count = %d \n", outer);
    printf ("Inner loop count = %d \n", inner);
  }
```

3) <u>Insertion Sort</u>

```
 #include <stdio.h>
 void insertionsort (int a[], int n){
 int i, j, key;
 int outer =0, inner =0;
 for(i=1; i<n; i++){
    outer++;
    key= a[i];
    j= i-1;
    while (j>= 0 && a[j] > key ){
        inner ++;
        a[j+1]=a[j];
        j--;
    }
   a[j+1]= key;
 }
 printf ("Outer Loop count = %d \n", outer);
 printf ("Inner Loop count = %d \n", inner);
}
```

Input used:

- Best case: Already sorted array. Ex: 1,2,3,4,5
- Average case: Random order array. Ex: 3,1,4,5,2
- Worst case: Reverse sorted array. Ex: 5,4,3,2,1

Iteration counts:

Bubble sort:

Case	Outer loop	Inner loop
Best	4	10
Average	4	10
Worst	4	10

Selection sort:

Case	Outer loop	Inner loop
Best	4	10
Average	4	10
Worst	4	10

Insertion sort:

Case	Outer loop	Inner loop
Best	4	0
Average	4	5
Worst	4	10

Time Complexity

- Bubble Sort – $O(n^2)$
- Selection Sort – $O(n^2)$
- Insertion Sort – Best case: $O(n)$
  Average & Worst case: $O(n^2)$

Conclusion

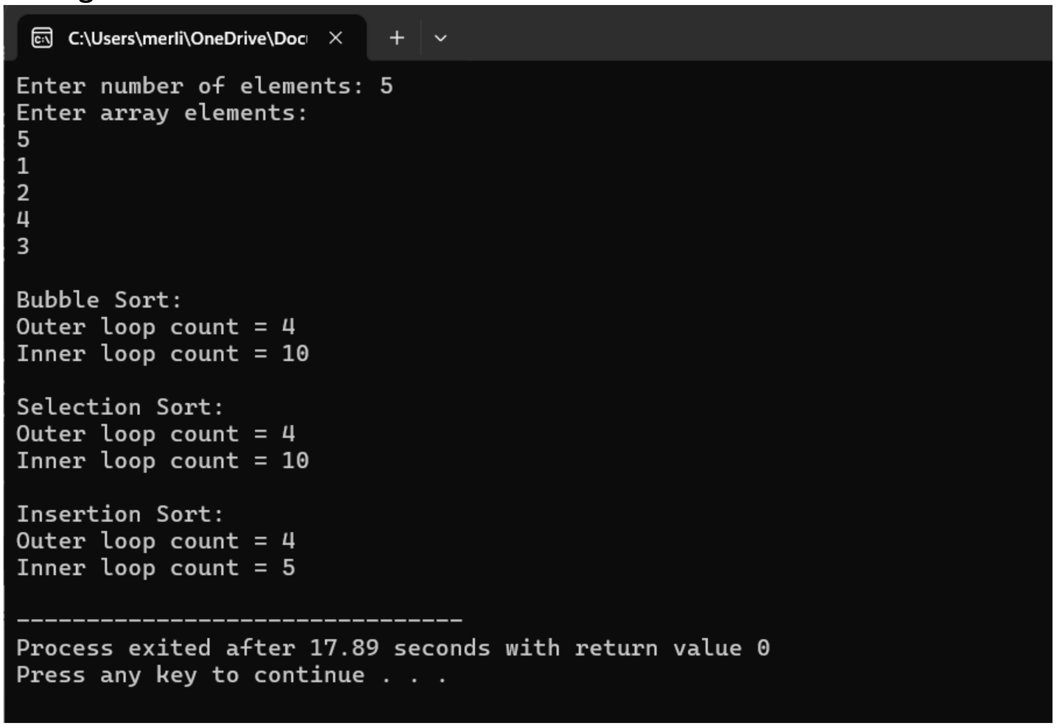
Bubble sort and Selection Sort perform the same number of iterations regardless of input order. Insertion Sort is more efficient when the array is already sorted, making it suitable for small or nearly sorted datasets. All three algorithms have quadratic time complexity in the worst case.

Merlin Xavier-44

Code:
```
#include <stdio.h>
/* -------- BUBBLE SORT -------- */
void bubblesort(int a[], int n) {
  int i, j, temp;
  int outer = 0, inner = 0;
  for(i = 0; i < n-1; i++) {
    outer++;
    for(j = 0; j < n-i-1; j++) {
      inner++;
      if(a[j] > a[j+1]) {
        temp = a[j];
        a[j] = a[j+1];
        a[j+1] = temp;
      }
    }
  }
  printf("\nBubble Sort:");
  printf("\nOuter loop count = %d", outer);
  printf("\nInner loop count = %d\n", inner);
}
/* -------- SELECTION SORT -------- */
void selectionsort(int a[], int n) {
  int i, j, min, temp;
  int outer = 0, inner = 0;
  for(i = 0; i < n-1; i++) {
    outer++;
    min = i;
    for(j = i+1; j < n; j++) {
      inner++;
      if(a[j] < a[min])
        min = j;
    }
    temp = a[i];
    a[i] = a[min];
    a[min] = temp;
  }
  printf("\nSelection Sort:");
  printf("\nOuter loop count = %d", outer);
  printf("\nInner loop count = %d\n", inner);
}
/* -------- INSERTION SORT -------- */
void insertionsort(int a[], int n) {
  int i, j, key;
  int outer = 0, inner = 0;
  for(i = 1; i < n; i++) {
    outer++;
    key = a[i];
    j = i - 1;
```

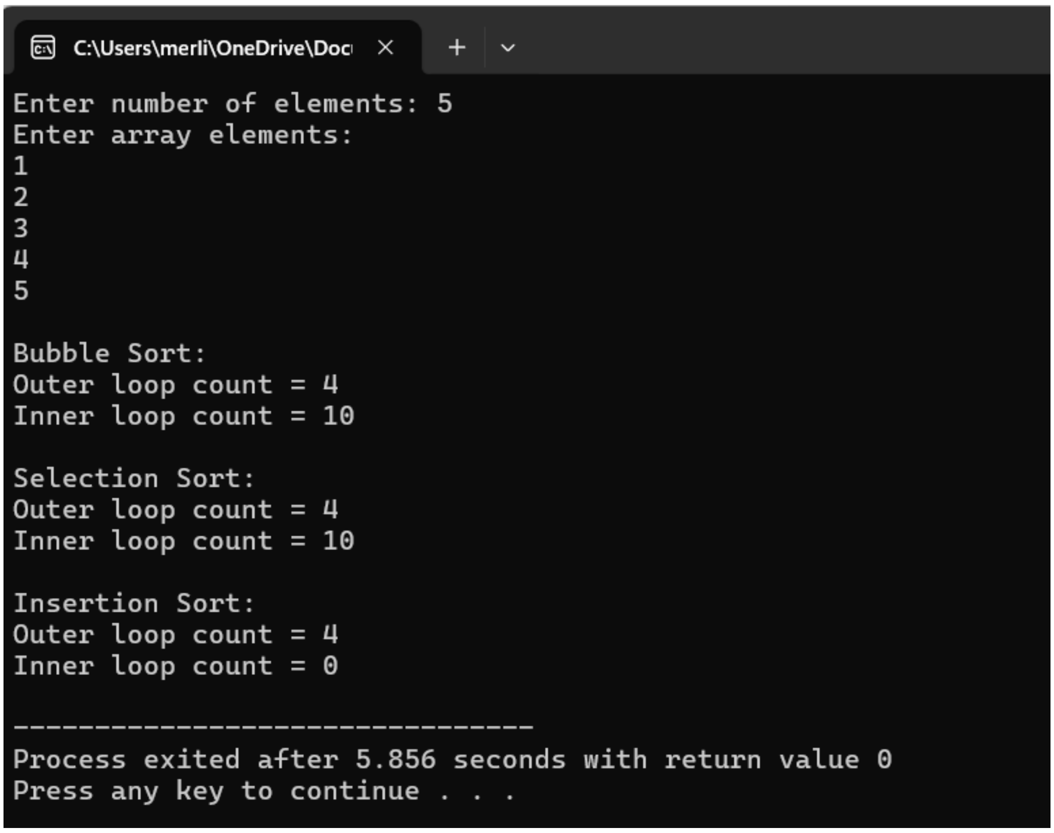
```
        while(j >= 0 && a[j] > key) {
            inner++;
            a[j + 1] = a[j];
            j--;
        }
        a[j + 1] = key;
    }
    printf("\nInsertion Sort:");
    printf("\nOuter loop count = %d", outer);
    printf("\nInner loop count = %d\n", inner);
}
/* -------- MAIN -------- */
int main() {
    int n, i;
    printf("Enter number of elements: ");
    scanf("%d", &n);
    int a[n], b[n], c[n];
    printf("Enter array elements:\n");
    for(i = 0; i < n; i++) {
        scanf("%d", &a[i]);
        b[i] = a[i];
        c[i] = a[i];
    }
    bubblesort(a, n);
    selectionsort(b, n);
    insertionsort(c, n);
    return 0;
}
```

Average Case:



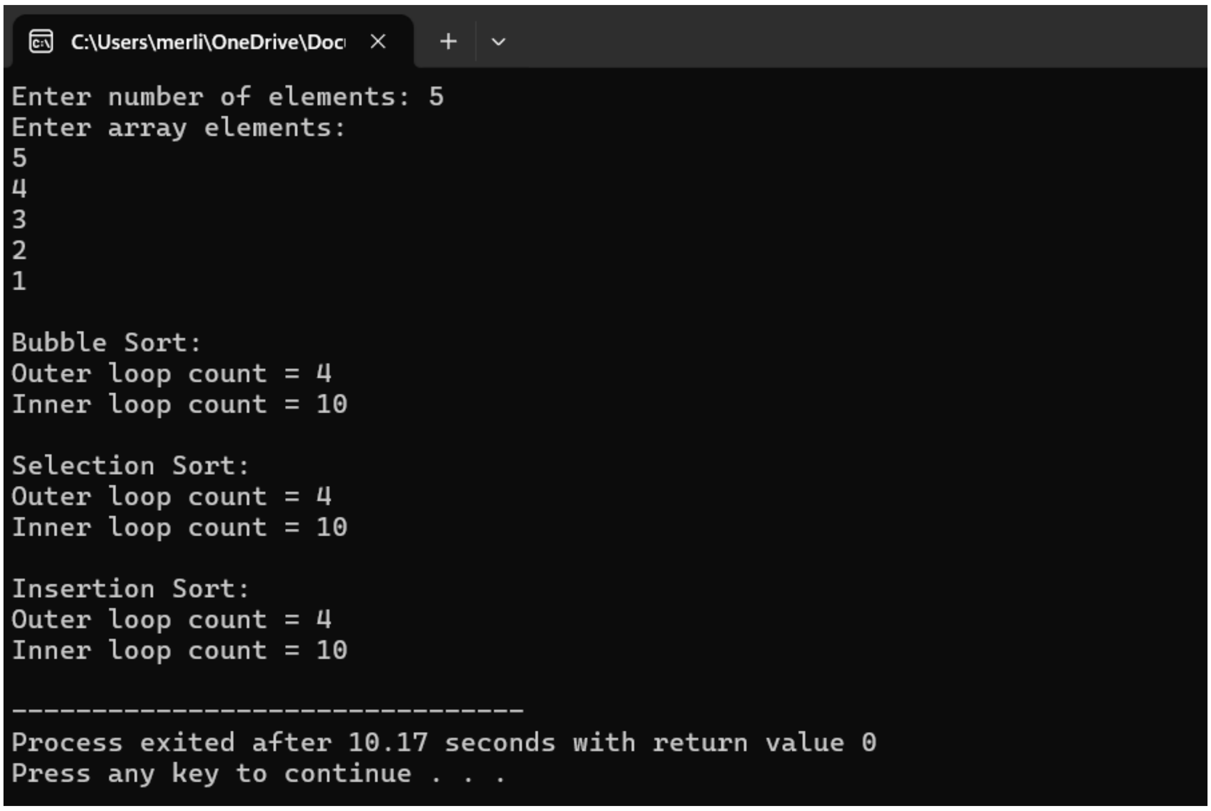
```
Enter number of elements: 5
Enter array elements:
5
1
2
4
3

Bubble Sort:
Outer loop count = 4
Inner loop count = 10

Selection Sort:
Outer loop count = 4
Inner loop count = 10

Insertion Sort:
Outer loop count = 4
Inner loop count = 5

--------------------------------
Process exited after 17.89 seconds with return value 0
Press any key to continue . . .
```

Best Case:



```
Enter number of elements: 5
Enter array elements:
1
2
3
4
5

Bubble Sort:
Outer loop count = 4
Inner loop count = 10

Selection Sort:
Outer loop count = 4
Inner loop count = 10

Insertion Sort:
Outer loop count = 4
Inner loop count = 0

--------------------------------
Process exited after 5.856 seconds with return value 0
Press any key to continue . . .
```

Worst Case:



```
Enter number of elements: 5
Enter array elements:
5
4
3
2
1

Bubble Sort:
Outer loop count = 4
Inner loop count = 10

Selection Sort:
Outer loop count = 4
Inner loop count = 10

Insertion Sort:
Outer loop count = 4
Inner loop count = 10

--------------------------------
Process exited after 10.17 seconds with return value 0
Press any key to continue . . .
```