AMRITA SCHOOL OF COMPUTING

DESIGN AND ANALYSIS OF ALGORITHMS

(23CSE211)

Name: Y.V.S.Likesh

Roll No.: CH.SC.U4CSE24152

Class: BTech (CSE-B)

School: Amrita School of Computing,
Chennai Campus.

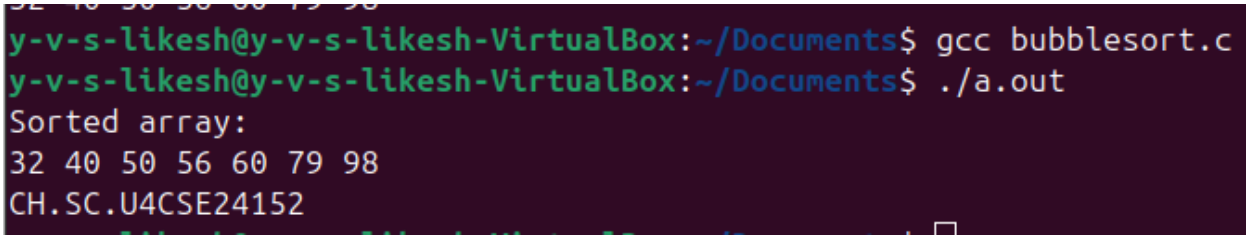
LAB-2

1) Bubble Sort

Code:

```
#include<stdbool.h>
#include<stdio.h>
void swap(int* x, int* y){
    int temp = *x;
    *x = *y;
    *y = temp;
}
void bubbleSort(int arr[], int n){
    int i,j;
    bool swapped;
    for(i=0;i<=n-1;i++){
        swapped = false;
     for(j=0;j<=n-1;j++){
      if(arr[j]>arr[j+1]){
            swap(&arr[j], &arr[j + 1]);
            swapped = true;
      }
    }
    if(swapped == false){
      break;
      }
    }
}
void printArray(int arr[], int size){
    int i;
    for (i = 0; i < size; i++)
        printf("%d ", arr[i]);
}
int main(){
    int arr[]={50,32,56,40,60,79,98};
    int n = sizeof(arr)/sizeof(arr[0]);
    bubbleSort(arr, n);
    printf("Sorted array: \n");
    printArray(arr, n);
    printf("\nCH.SC.U4CSE24152\n");
}
```

Output:



Space Complexity:

The space complexity of this program is O(1) - constant space. Because the program uses a fixed number of variables, independent of the input size, its space usage does not grow.

Time Complexity:

The time complexity of this program is O(n^2). Because of Two nested loops compare adjacent elements.

2) Insertion Sort

Code:

```c
#include<stdio.h>
void insertionSort(int arr[], int n){
    for (int i=1;i<=n-1;i++){
        int key = arr[i];
        int j = i-1;
        while(j>=0 && arr[j]>key){
            arr[j+1] = arr[j];
            j = j-1;
        }
        arr[j+1] = key;
    }
}
void printArray(int arr[], int n){
    for(int i=0;i<n;++i){
        printf("%d ",arr[i]);
    }
    printf("\n");
}
int main(){
    int arr[] = {19,20,12,5,6,11,19};
    int n = sizeof(arr)/sizeof(arr[0]);
    insertionSort(arr, n);
    printArray(arr, n);
    printf("CH.SC.U4CSE24152\n");
    return 0;
}
```

Output:

```
y-v-s-likesh@y-v-s-likesh-VirtualBox:~/Documents$ gcc insertionsort.c
y-v-s-likesh@y-v-s-likesh-VirtualBox:~/Documents$ ./a.out
5 6 11 12 19 19 20
CH.SC.U4CSE24152
```

Space Complexity:

The space complexity of this program is O(1) - constant space. Because there is no recursion, no arrays, no dynamic memory allocation, the amount of memory used does not depend on n.

Time Complexity:

The time complexity of this program is O(n^2). Because in worst case, the array is reverse sorted.

3) Selection Sort

Code:

```c
#include<stdio.h>
void selectionSort(int arr[],int n){
    for(int i=0;i<=n-1;i++){
      int min_idx = i;
      for(int j =i;j<=n-1;j++){
        if(arr[j] < arr[min_idx]){
            min_idx = j;
        }
      }
      int temp = arr[i];
      arr[i] = arr[min_idx];
      arr[min_idx] = temp;
  }
}
void printArray(int arr[],int n){
    for(int i=0;i<=n-1;i++){
       printf("%d ", arr[i]);
    }
    printf("\n");
}
int main(){
    int arr[]={64,32,45,68,19,28,71};
    int n = sizeof(arr)/sizeof(arr[0]);
    printf("Original array: ");
    printArray(arr,n);
    selectionSort(arr,n);
    printf("Sorted array: ");
    printArray(arr,n);
    printf("CH.SC.U4CSE24152\n");
    return 0;
}
```

Output:

```
y-v-s-likesh@y-v-s-likesh-VirtualBox:~/Documents$ gcc selectionsort.c
y-v-s-likesh@y-v-s-likesh-VirtualBox:~/Documents$ ./a.out
Original array: 64 32 45 68 19 28 71
Sorted array: 19 28 32 45 64 68 71
CH.SC.U4CSE24152
```

Space Complexity:

The space complexity of this program is O(1) - constant space. Because there is no recursion, no arrays, no dynamic memory allocation, the amount of memory used does not depend on n.

Time Complexity:

The time complexity of this program is O(n^2). Because it always compares all remaining elements to find the minimum.

4) Bucket Sort

Code:

```
#include<stdio.h>
#include<stdlib.h>
typedef struct Node{
    float data;
    struct Node* next;
}Node;
void insert(Node** bucket, float value){
    Node* newNode = (Node*)malloc(sizeof(Node));
    newNode->data = value;
    newNode->next = NULL;
    if(*bucket == NULL || (*bucket)->data >= value){
      newNode->next = *bucket;
      *bucket = newNode;
    }
    else{
      Node* current = *bucket;
      while(current->next && current->next->data < value){
        current = current->next;
      }
      newNode->next = current->next;
      current->next = newNode;
    }
}
void bucketSort(float arr[], int n){
    Node* buckets[n];
    for(int i=0;i<n;i++){
      buckets[i] = NULL;
    }
    for(int i=0;i<n;i++){
        int index = arr[i]*n;
        if(index == n){
          index = n - 1;
        }
        insert(&buckets[index], arr[i]);
    }
    int idx = 0;
    for(int i=0;i<n;i++){
      Node* node = buckets[i];
      while(node){
        arr[idx++] = node->data;
        node = node->next;
      }
    }
}
```

```c
int main(){
    int n;
    printf("Enter number of elements: ");
    scanf("%d", &n);
    float arr[n];
    printf("Enter floating point numbers between 0 and 1:\n");
    for(int i=0;i<n;i++){
      scanf("%f", &arr[i]);
    }
    bucketSort(arr, n);
    printf("Sorted array:\n");
    for(int i=0;i<n;i++){
        printf("%f ", arr[i]);
    }
    printf("\nCH.SC.U4CSE24152\n");
    return 0;
}
```

Output:

```
y-v-s-likesh@y-v-s-likesh-VirtualBox:~/Documents$ gcc bucketsort.c
y-v-s-likesh@y-v-s-likesh-VirtualBox:~/Documents$ ./a.out
Enter number of elements: 3
Enter floating point numbers between 0 and 1:
0.1 0.6 0.2
Sorted array:
0.100000 0.200000 0.600000
CH.SC.U4CSE24152
```

Space Complexity:

The space complexity of this problem is O(n). There are no arrays or dynamic memory allocation, but the recursive call stack makes the memory usage depend on n, resulting in O(n) space complexity.

Time Complexity:

The time complexity of this program is O(n^2). Because elements are distributed into n buckets. In worst case, All elements in one bucket and behaves insertion sort.

5) Heap Sort

Code:

```c
#include <stdio.h>
void heapify(int arr[],int n,int i){
    int largest = i;
    int l = 2*i+1;
    int r = 2*i+2;
    if(l<n && arr[l]>arr[largest]){
        largest = l;
    }
    if(r<n && arr[r]>arr[largest]){
        largest = r;
    }
    if(largest != i){
        int temp = arr[i];
        arr[i] = arr[largest];
        arr[largest] = temp;
        heapify(arr,n,largest);
    }
}
void heapSort(int arr[],int n){
    for(int i=n/2-1;i>=0;--i){
        heapify(arr, n, i);
    }
    for(int i=n-1;i>0;i--){
        int temp = arr[0];
        arr[0] = arr[i];
        arr[i] = temp;
        heapify(arr,i,0);
    }
}
int main(){
    int arr[] = {15,16,19,20,26,10,3,18};
    int n = sizeof(arr)/sizeof(arr[0]);
    heapSort(arr, n);
    for(int i=0;i<n;++i){
        printf("%d ",arr[i]);
    }
    printf("\nCH.SC.U4CSE24152\n");
    return 0;
}
```

Output:

```
y-v-s-likesh@y-v-s-likesh-VirtualBox:~/Documents$ gcc heapsort.c
y-v-s-likesh@y-v-s-likesh-VirtualBox:~/Documents$ ./a.out
3 10 15 16 18 19 20 26
CH.SC.U4CSE24152
```

Space Complexity:

The space complexity of this program is O(1) - constant space. Because the memory used does not grow with input size, the total space remains constant.

Time Complexity:

The time complexity of this program is O(n log n). Because of heap construction and need to perform n times.