Design Analysis and Algorithm – Lab Work

Week 6

Question 1:Write a Program to perform Quick Sort using first element, middle element and random element.

Code:

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

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

int partition_first(int a[],int low,int high){
    int pivot=a[low];
    int i=low+1,j=high;
    while(i<=j){
        while(i<=high && a[i]<=pivot) i++;
        while(a[j]>pivot) j--;
        if(i<j) swap(&a[i],&a[j]);
    }
    swap(&a[low],&a[j]);
    return j;
}

int partition_last(int a[],int low,int high){
    int pivot=a[high];
    int i=low-1;
    for(int j=low;j<high;j++){
        if(a[j]<pivot){
            i++;
            swap(&a[i],&a[j]);
        }
    }
    swap(&a[i+1],&a[high]);
    return i+1;
}

int partition_random(int a[],int low,int high){
    int r=low+rand()%(high-low+1);
    swap(&a[r],&a[high]);
    return partition_last(a,low,high);
```

```c
}

void quick_first(int a[],int low,int high){
    if(low<high){
        int p=partition_first(a,low,high);
        quick_first(a,low,p-1);
        quick_first(a,p+1,high);
    }
}

void quick_last(int a[],int low,int high){
    if(low<high){
        int p=partition_last(a,low,high);
        quick_last(a,low,p-1);
        quick_last(a,p+1,high);
    }
}

void quick_random(int a[],int low,int high){
    if(low<high){
        int p=partition_random(a,low,high);
        quick_random(a,low,p-1);
        quick_random(a,p+1,high);
    }
}

int main(){
    printf("ch.sc.u4cse24149\n");
    int n,choice;
    scanf("%d",&n);
    int a[n];
    for(int i=0;i<n;i++) scanf("%d",&a[i]);
    scanf("%d",&choice);
    srand(time(0));
    if(choice==1) quick_first(a,0,n-1);
    else if(choice==2) quick_last(a,0,n-1);
    else if(choice==3) quick_random(a,0,n-1);
    for(int i=0;i<n;i++) printf("%d ",a[i]);
    return 0;
}
```

Output:

```
E:\Vighranth SK>gcc CH.SC.U4CSE24149.c

E:\Vighranth SK>a
CH.SC.U4CSE24149
12
157 110 147 122 111 149 151 141 123 112 117 133
1
110 111 112 117 122 123 133 141 147 149 151 157
```

```
E:\Vighranth SK>gcc CH.SC.U4CSE24149.c

E:\Vighranth SK>A
CH.SC.U4CSE24149
12
157 110 147 122 111 149 151 141 123 112 117 133
2
110 111 112 117 122 123 133 141 147 149 151 157
E:\Vighranth SK>gcc CH.SC.U4CSE24149.c

E:\Vighranth SK>A
CH.SC.U4CSE24149
12
157 110 147 122 111 149 151 141 123 112 117 133
3
110 111 112 117 122 123 133 141 147 149 151 157
```

```
E:\Vighranth SK>gcc CH.SC.U4CSE24149.c

E:\Vighranth SK>A
CH.SC.U4CSE24149
12
157 110 147 122 111 149 151 141 123 112 117 133
2
110 111 112 117 122 123 133 141 147 149 151 157
E:\Vighranth SK>gcc CH.SC.U4CSE24149.c

E:\Vighranth SK>A
CH.SC.U4CSE24149
12
157 110 147 122 111 149 151 141 123 112 117 133
3
110 111 112 117 122 123 133 141 147 149 151 157
```

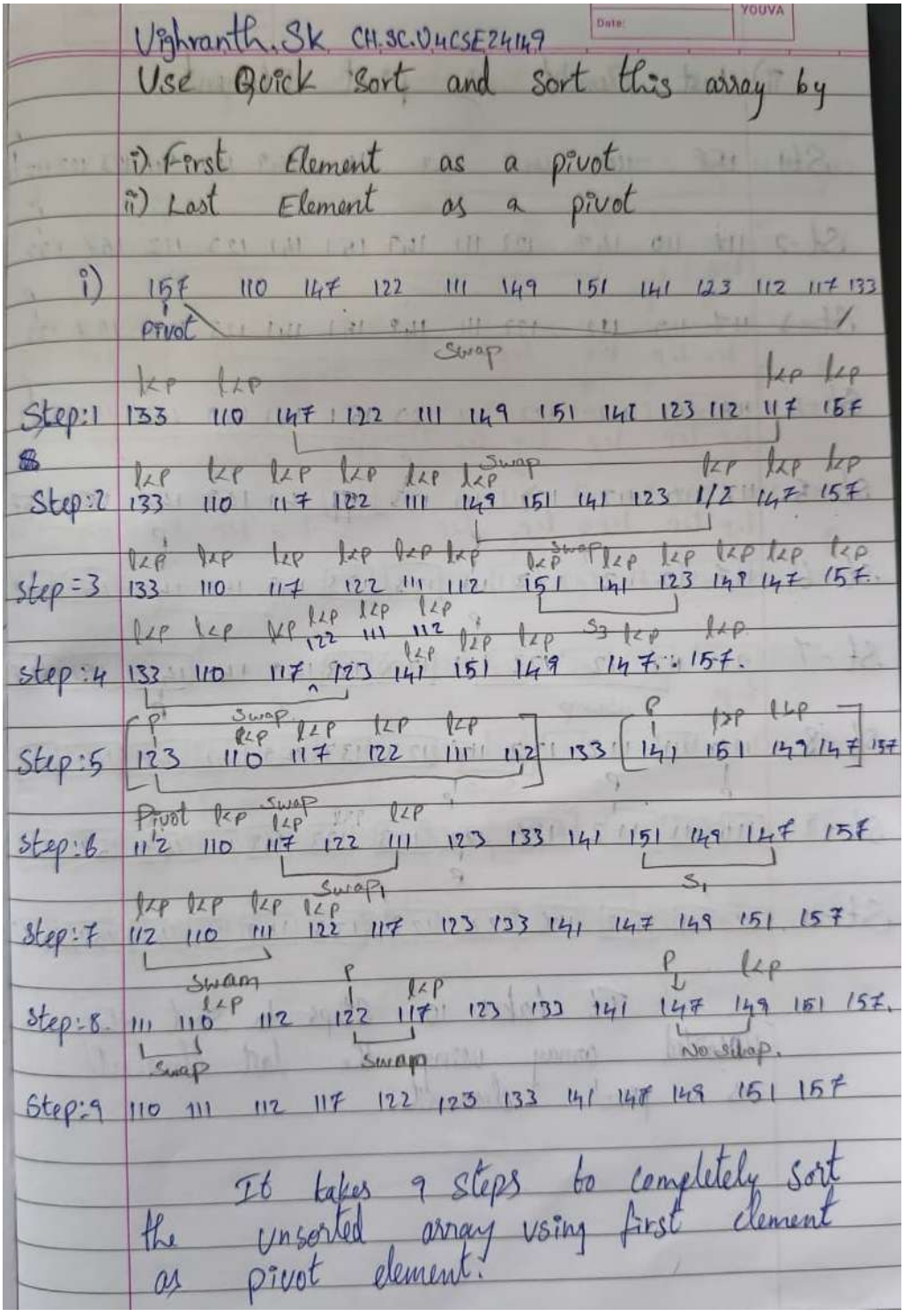
Space Complexity (Worst Case):

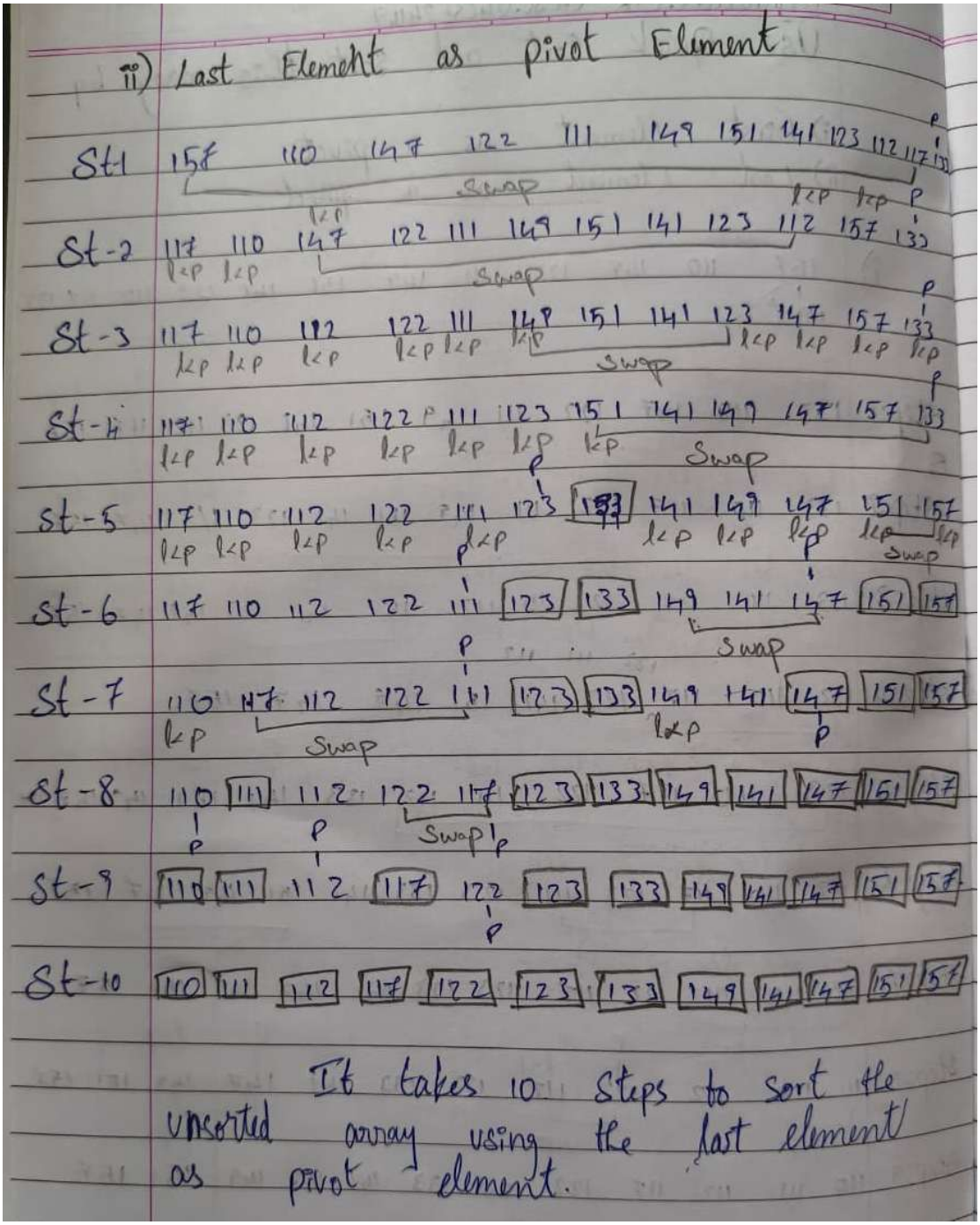
In the worst case, the program stores a single integer array `a[n]`. Since each integer occupies 4 bytes, the total memory required to store the array is `n × 4 = 4n` bytes. Apart from this, the program uses a few integer variables such as `n`, `choice`, loop counters, pivot and index variables, and a temporary variable used for swapping, all of which together occupy constant memory. Quick Sort is a recursive algorithm, and in the worst case the recursion depth becomes `n` due to highly unbalanced partitions. Hence, the recursion stack requires `O(n)` space. Therefore, the worst case **Space Complexity** of the program is **O(n)**.

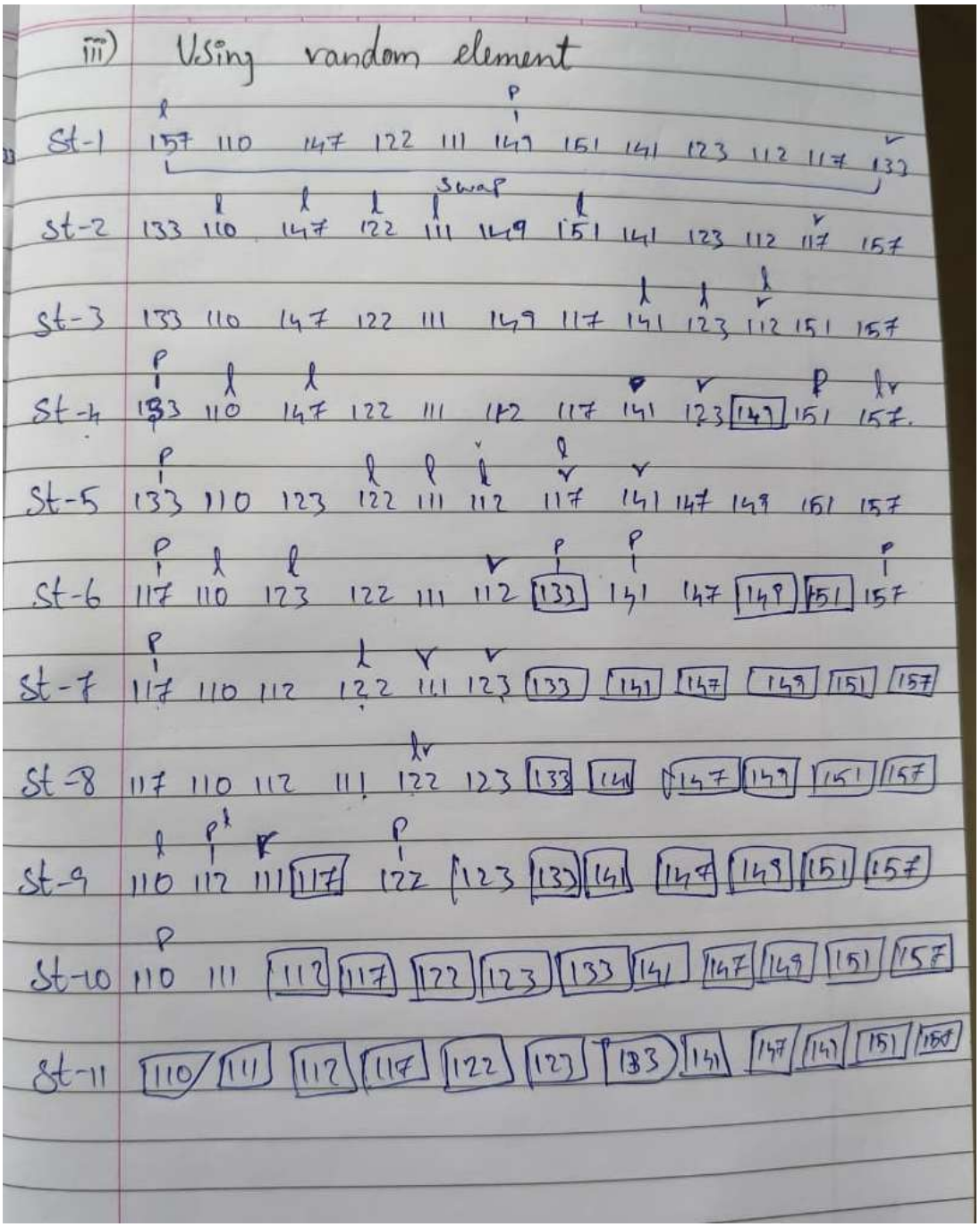
Time Complexity (Worst Case):

In the worst case, the pivot selection (first element pivot, last element pivot, or random pivot in rare cases) always produces highly unbalanced partitions where one subarray contains `n-1` elements and the other contains `0` elements. In such a situation, the partition function performs `O(n)` comparisons in the first call, `O(n-1)` in the next call, and so on until `O(1)`. Hence, the total number of comparisons becomes
`n + (n-1) + (n-2) + ... + 1 = n(n-1)/2`,
which is proportional to n^2. Therefore, the worst case **Time Complexity** of the program is **$O(n^2)$**.

Working:


Vighranth.Sk CH.SC.U4CSE24149
Use Quick Sort and sort this array by
i) First Element as a pivot
ii) Last Element as a pivot
i) 157 110 147 122 111 149 151 141 123 112 117 133
pivot
Swap
Step:1 133 110 147 122 111 149 151 141 123 112 117 157
Step:2 133 110 117 122 111 149 151 141 123 112 147 157
step=3 133 110 117 122 111 112 151 141 123 149 147 157.
step:4 133 110 117 123 141 151 149 147 157.
Step:5 123 110 117 122 111 112 133 141 151 149 147 157
Step:6 112 110 117 122 111 123 133 141 151 149 147 157
Step:7 112 110 111 122 117 123 133 141 147 149 151 157
Step:8 111 110 112 122 117 123 133 141 147 149 151 157.
No Swap.
Step:9 110 111 112 117 122 123 133 141 147 149 151 157
It takes 9 steps to completely sort the unsorted array using first element as pivot element.


ii) Last Element as pivot Element
St-1
St-2
St-3
St-4
St-5
St-6
St-7
St-8
St-9
St-10
Swap
It takes 10 Steps to sort the unsorted array using the last element as pivot element.


iii) Using random element
St-1 157 110 147 122 111 149 151 141 123 112 117 133
swap
St-2 133 110 147 122 111 149 151 141 123 112 117 157
St-3 133 110 147 122 111 149 117 141 123 112 151 157
St-4 133 110 147 122 111 112 117 141 123 149 151 157
St-5 133 110 123 122 111 112 117 141 147 149 151 157
St-6 117 110 123 122 111 112 133 141 147 149 151 157
St-7 117 110 112 122 111 123 133 141 147 149 151 157
St-8 117 110 112 111 122 123 133 141 147 149 151 157
St-9 110 112 111 117 122 123 133 141 147 149 151 157
St-10 110 111 112 117 122 123 133 141 147 149 151 157
St-11 110 111 112 117 122 123 133 141 147 149 151 157