AMRITA VISHWA VIDYAPEETHAM

Amrita School of Computing

23CSE211

Design and Analysis of Algorithms

WEEK – 5 (22 January 2026)

Susendran M

CH.SC.U4CSE24154

BACHELOR OF TECHNOLOGY

IN

COMPUTER SCIENCE AND ENGINEERING

AMRITA VISHWA VIDYAPEETHAM

AMRITA SCHOOL OF COMPUTING

CHENNAI

Quick Sorting :

Quick Sort is a Divide and Conquer sorting algorithm.
A pivot element is selected (can be any element).
The array is partitioned such that elements less than the pivot are placed on the left and greater than the pivot on the right.
The pivot is placed in its correct position.
The same process is recursively applied to the left and right subarrays.
Sorting continues until all subarrays have one or no elements.

Code :

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
void swap(int *a, int *b)
{
   int temp = *a;
   *a = *b;
   *b = temp;
}
int hoarePartition(int arr[], int low, int high)
{
   int pivot = arr[low];
   int i = low - 1;
   int j = high + 1;
   while (1)
        {
      do
                {
         i++;
      }
                while (arr[i] < pivot);
      do
                {
         j--;
      }
                while (arr[j] > pivot);
      if (i >= j)
      {
         return j;
      }
```

```
        swap(&arr[i], &arr[j]);
    }
}
void quickSort(int arr[], int low, int high)
{
    if (low < high)
        {
        int p = hoarePartition(arr, low, high);
        quickSort(arr, low, p);
        quickSort(arr, p + 1, high);
    }
}
void choosePivot(int arr[], int low, int high, int choice)
{
    int pivotIndex;
    if (choice == 1)
        {
        return;
    }
    else if (choice == 2)
        {
        swap(&arr[low], &arr[high]);
    }
    else if (choice == 3)
        {
        pivotIndex = low + rand() % (high - low + 1);
        swap(&arr[low], &arr[pivotIndex]);
    }
}
void printArray(int arr[], int n)
{
    for (int i = 0; i < n; i++)
    {
        printf("%d ", arr[i]);
    }
    printf("\n");
}
int main()
{
    int arr[] = {157,110,147,122,111,149,151,141,123,112,117,133};
    int n = sizeof(arr) / sizeof(arr[0]);
    int choice;
    srand(time(NULL));
    printf("Choose Pivot:\n");
```

```
    printf("1. First Element\n");
    printf("2. Last Element\n");
    printf("3. Random Element\n");
    printf("Enter choice: ");
    scanf("%d", &choice);
    choosePivot(arr, 0, n - 1, choice);
    quickSort(arr, 0, n - 1);
    printf("Sorted Array:\n");
    printArray(arr, n);
    return 0;
}
```

Output :

```
Choose Pivot:
1. First Element
2. Last Element
3. Random Element
Enter choice: 3
Sorted Array:
110 111 112 117 122 123 133 141 147 149 151 157

--------------------------------
Process exited after 1.714 seconds with return value 0
Press any key to continue . . .
```

1. First Element :

Step 1: Choose Pivot
Take the first element of the (sub)array as the pivot.

Step 2: Initialize Pointers
Set

$i = low - 1$

$j = high + 1$

Step 3: Move Left Pointer (i)
Move i from left to right
Stop when an element greater than or equal to pivot is found

Step 4: Move Right Pointer (j)
Move j from right to left
Stop when an element less than or equal to pivot is found

Step 5: Swap or Stop
If $i < j$
Swap the elements at positions i and j
If $i \geq j$
Stop partitioning and fix j as partition index

Step 6: Divide the Array
Split the array into two parts:

Left part: low to j

Right part: j + 1 to high

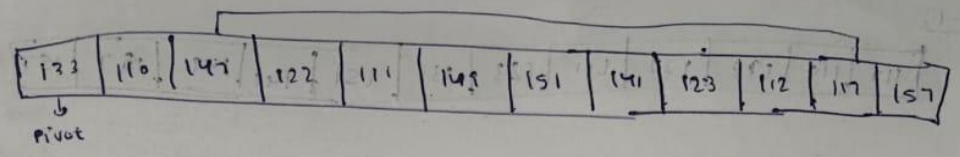
Step 7: Repeat Recursively
Apply the same steps to both subarrays
Stop when subarray size becomes 0 or 1

22.1.26

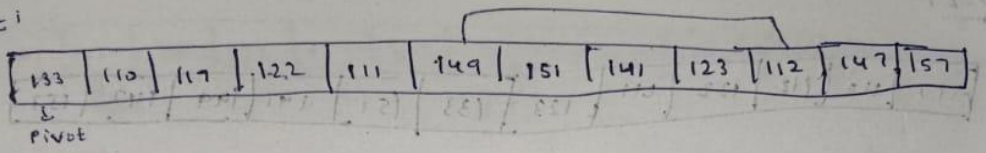
List = [157, 110, 147, 122, 111, 149, 151, 141, 123, 112, 117, 133]
↓ Pivot

Case - 1: (First Element)

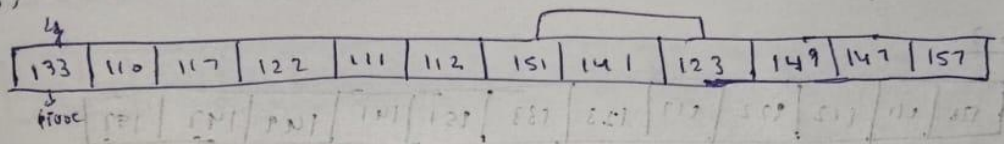
Step-1 ;



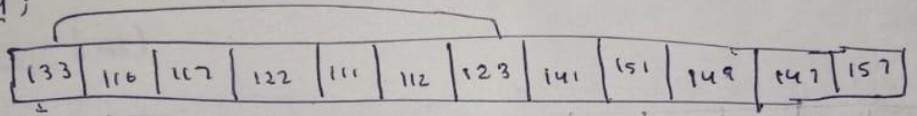
Step-2 ;



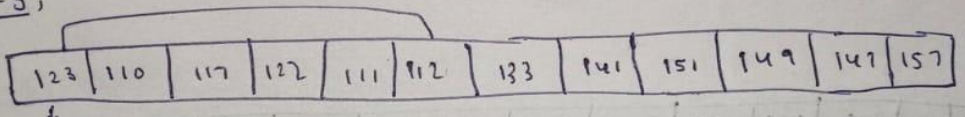
Step-3 ;



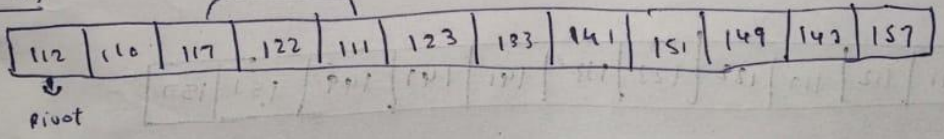
Step-4 ;



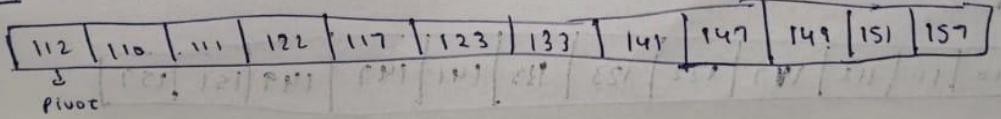
Step-5 ;



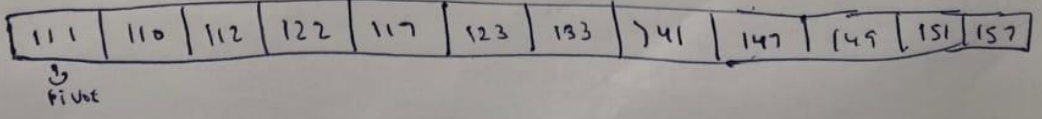
Step-6 ;



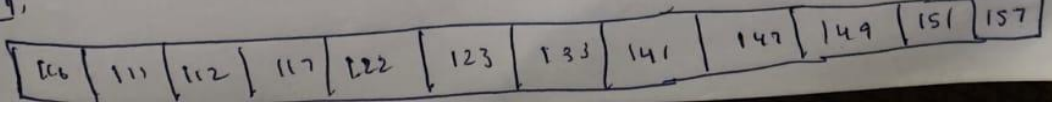
Step-7 ;



Step-8 ;



Step-9 ;



2. Last Element :

Step 1: Choose Pivot
Take the last element of the (sub)array as the pivot.

Step 2: Initialize Index
Set
$i = low - 1$

Step 3: Scan the Array
For each element from index low to high − 1:
If element ≤ pivot:
Increment i
Swap element at index i with current element

Step 4: Place Pivot
Swap pivot (last element) with element at index i + 1

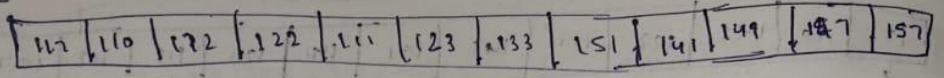
Step 5: Partition the Array
Pivot is now at its correct sorted position
Left subarray: elements ≤ pivot
Right subarray: elements > pivot

Step 6: Repeat Recursively
Apply the same steps to left and right subarrays
Stop when subarray size is 0 or 1

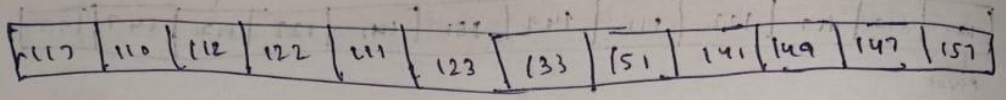
Case - 2 : (Last Element)

arr[] = [157, 110, 147, 122, 111, 149, 151, 141, 123, 112, 117, 133]
pivot

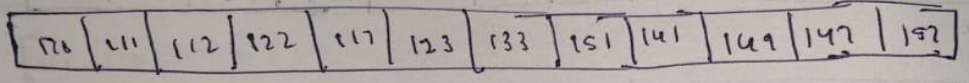
Step-1;



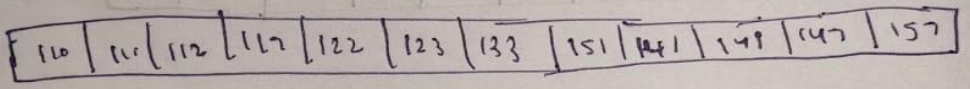
Step-2;



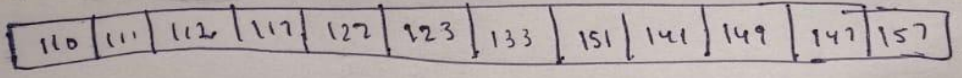
Step-3;



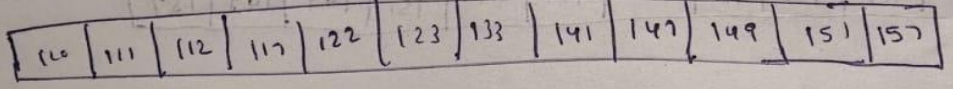
Step-4;



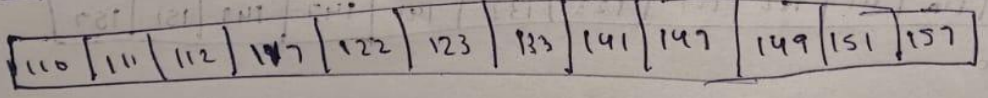
Step-5;



Step-6;



Step-7;



3. Random Element :

Step 1: Select Pivot
Randomly choose an index between low and high
Swap the chosen element with the first element

Step 2: Fix Pivot
Treat the first element as the pivot

Step 3: Initialize Pointers
Set
$i = low - 1$
$j = high + 1$

Step 4: Move Left Pointer (i)
Move i from left to right
Stop when an element greater than or equal to pivot is found

Step 5: Move Right Pointer (j)
Move j from right to left
Stop when an element less than or equal to pivot is found

Step 6: Swap or Stop
If $i < j$
Swap elements at positions i and j
If $i \geq j$
Stop partitioning and fix j as partition index

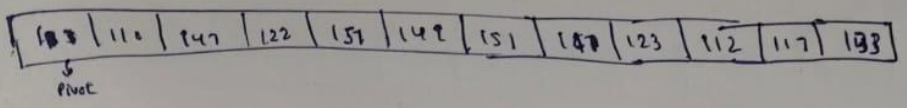
Step 7: Divide the Array
Left subarray: low to j
Right subarray: j + 1 to high

Step 8: Repeat Recursively
Apply the same steps to both subarrays
Stop when subarray size becomes 0 or 1

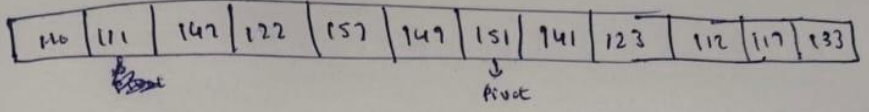
case-3: (Random element)

arr[] = [157, 110, 147, 122, 111, 149, 151, 141, 123, 112, 117, 133]

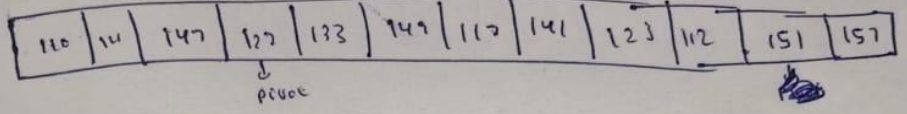
Step-1: (111)



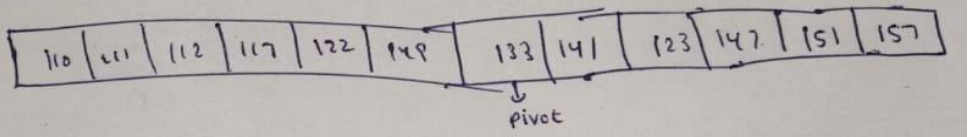
Step-2: (151)



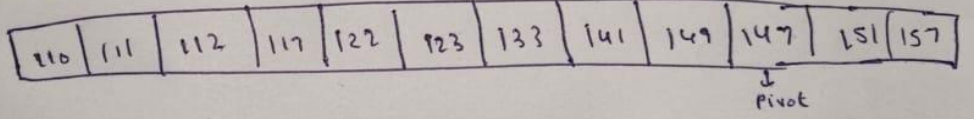
Step-3: (122)



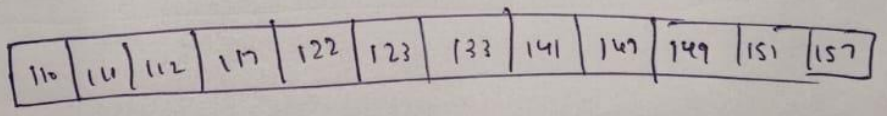
Step-4: (133)



Step-5: (147)



Step-6:



Inference :

Based on the sorting, Random pivot sorting is best compared to First and Last Pivoting.