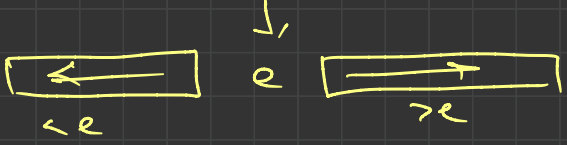
For: Step → element

↳ right place

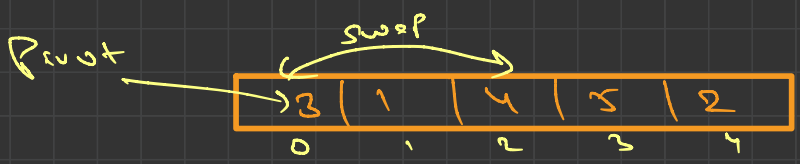


Quick Sort

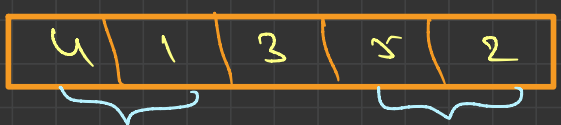
↳ Partition

↳ recursion

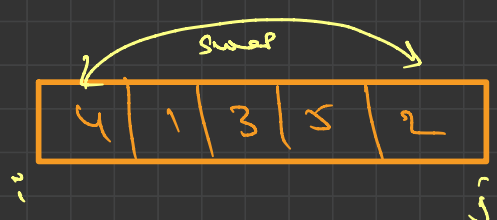
Partition



- take pivot element
- count all element < pivot
- pivot → place on index & ++ count



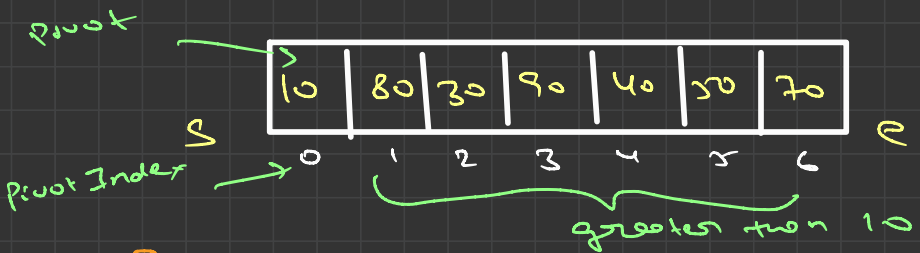
- use → <e | e | >e



2 1 3 5 4
i j

2 1 3 5 4
ij

```
int partition (vector<int> &arr, int s, int e)
{
    // get pivot
   int pivot = arr[s];

   // count elements less than or equal to pivot
   int count = 0;

   for(int i = s+1; i <= e; i++){
      if(arr[i] <= pivot){
         count++;
      }
   }

   int pivotIndex = s + count;

   // place pivot at right place

   swap(arr[s],arr[pivotIndex]);

   // use condition -> | < e | e | > e |

   int i = s, j = e;

   while(i < pivotIndex && j > pivotIndex){

      while(arr[i] <= pivot){
         i++;
      }

      while(arr[j] > pivot){
         j--;
      }

      if(i < pivotIndex && j > pivotIndex){
         swap(arr[i], arr[j]);
         i++;
         j--;
      }
   }
   return pivotIndex;
}

//Function to sort an array using quick sort
algorithm.
void solve(vector<int> &arr, int s, int e)
{
    // base case
    if(s >= e) return;

    // find pivot Index
    int p = partition(arr, s , e);

    // rec call
    solve(arr, s, p-1);
    solve(arr, p+1, e);
}

vector<int> quickSort(vector<int> arr)
{
    int s = 0;
    int e = arr.size() - 1;
    solve(arr, s, e);

    return arr;
}
```



Pivot = 10
count = 0

pivotIndex = s + count

↳ 0 + 0 ⇒ 0

while (i < pivotIndex && j > pivotIndex)

↳ return false

return pivotIndex ;

↳ 0

Solve (arr, p+1, e) ;

↳ index → 1 = s

pivot = 80

count = 0

↳ 4

pivotIndex = S + count ;

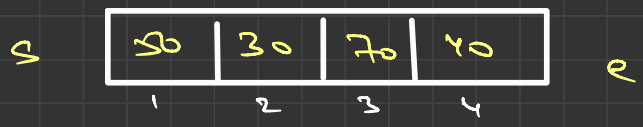
↳ 1 + 4 = 5 ;

Swap (arr[S] , arr[pivotIndex]) ;

return pivotIndex ;

Solve (arr, S, P-1) ;

Solve (arr, P+1 , e) ;

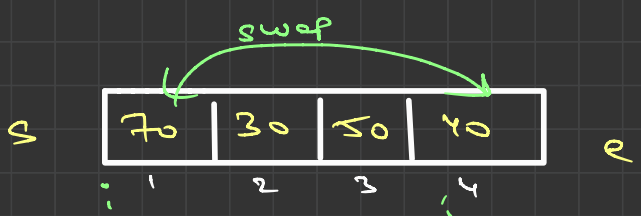


Pivot = 50
Count = 0
↳ 2

Pivot Index = s + count
↳ 1 + 2 ⇒ 3

Swap

Maintain the cond^n :- $<e \mid e \mid >e$



return pivot Index;
↳ 3

Solve (arr, s, p-1);
Solve (arr, p+1, e);