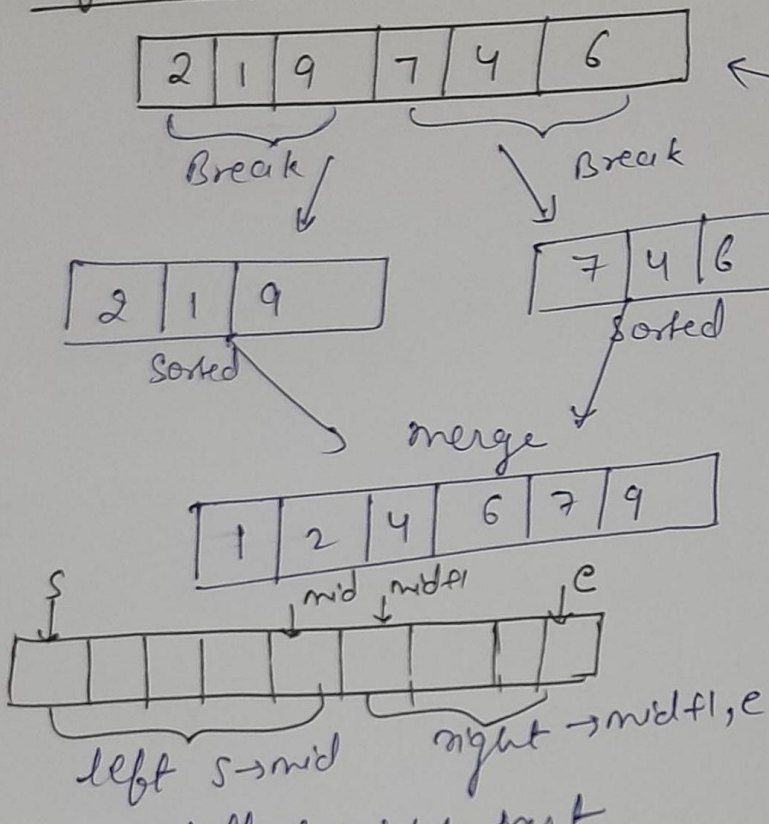
Lecture-22 DnC Class-1 Date - 18/10/23

Merge Sort



High level soln
i/p

I → find mid break

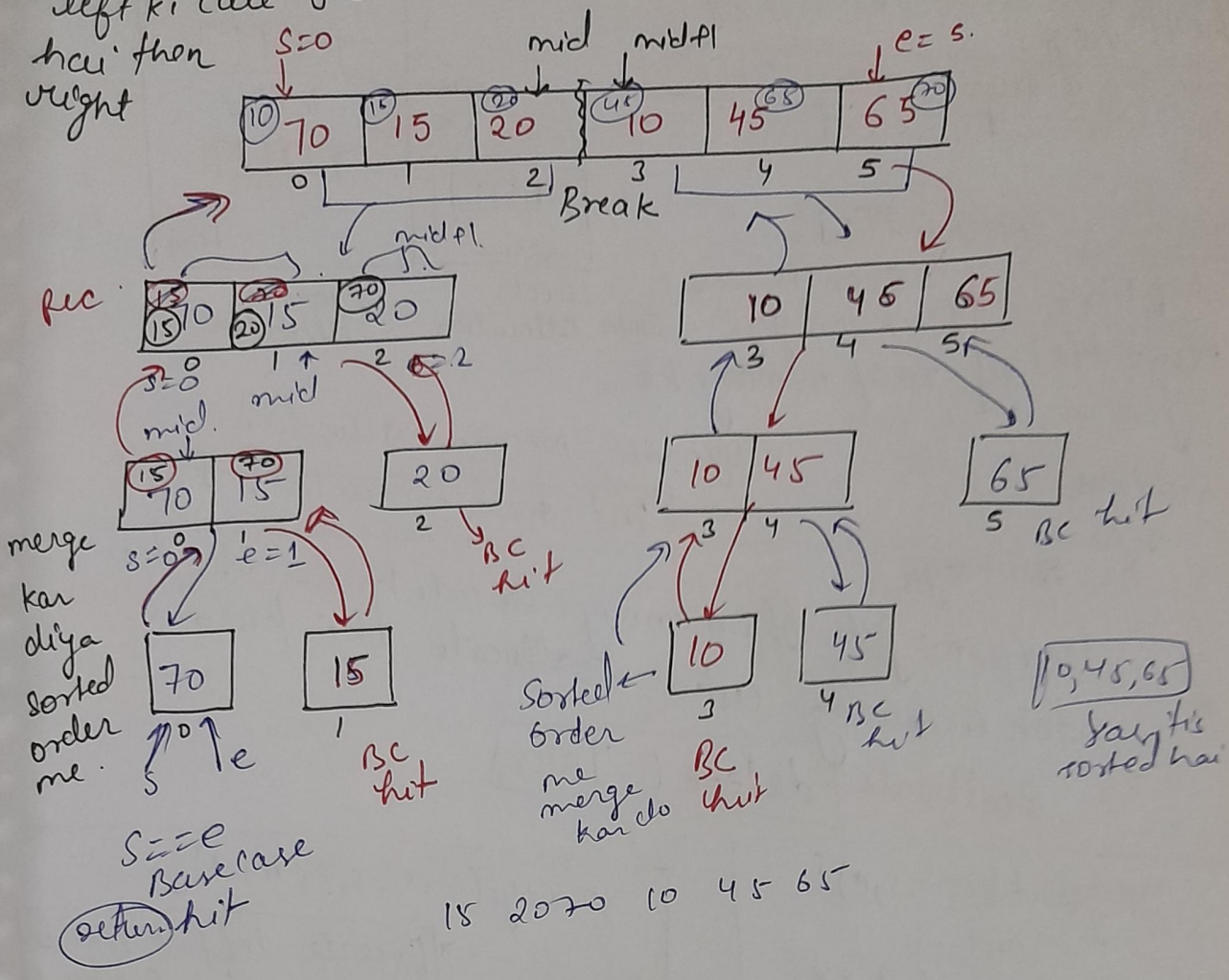
II Recursion apply karke sort kardo

III → Merge 2 sorted arrays.

break into left & right part
Recursion

```
mergesort(arr, s, e){
    //B.C
    if(s>=e)
        return;
    int mid = s+(e-s)/2 (s+e)/2;
    mergeSort(arr, s, mid);
    mergeSort(arr, mid+1, e);
    merge(arr, s, e, mid);
}
```

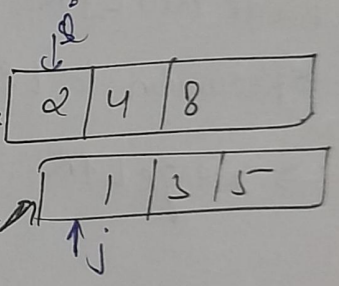
code me phele mid = $\frac{s+e}{2}$ = $\frac{0+5}{2}$ = 2.
left ki call jati hai then right

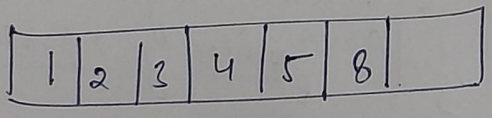


15 20 70 10 45 65

Ques Merge 2 sorted Array :-

i<j
j<i
1 chota hai ya 2 chota hai
1 then put (1).

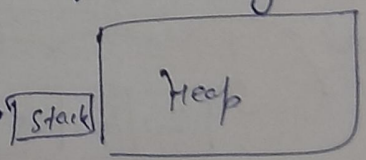




int arr[5] → fixed size
↳ static array

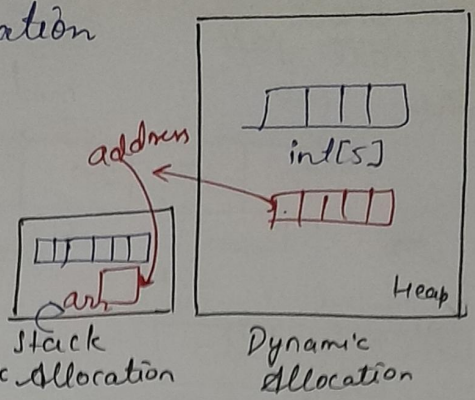
Jab bhi program banate hai toh stack memory or heap memory bnati hai

Heap ke upar agar memory allocate karwani hai toh (P/g) (new) keyword use karte hai.



heap pe Runtime pe to memory allocation hogi.

```
int *arr = new int[5]
int arr[5];
```



new int[5]; → 5 continuous block mil gye heap memory ke

```
int *ptr
= new int[5];
```

↓ address return kar rha hoga

Dynamic Memory Allocation

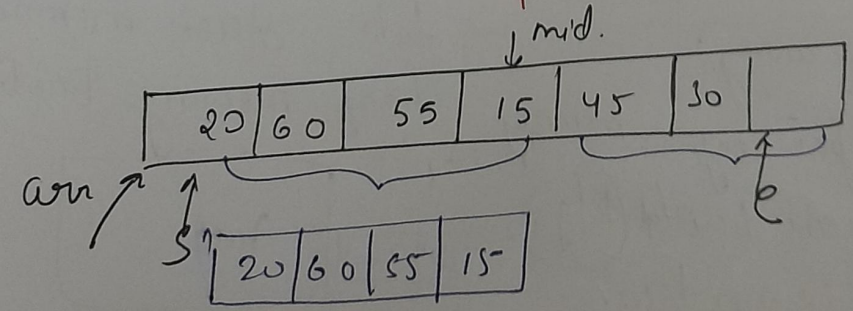
```
int *arr = new int[n];
```

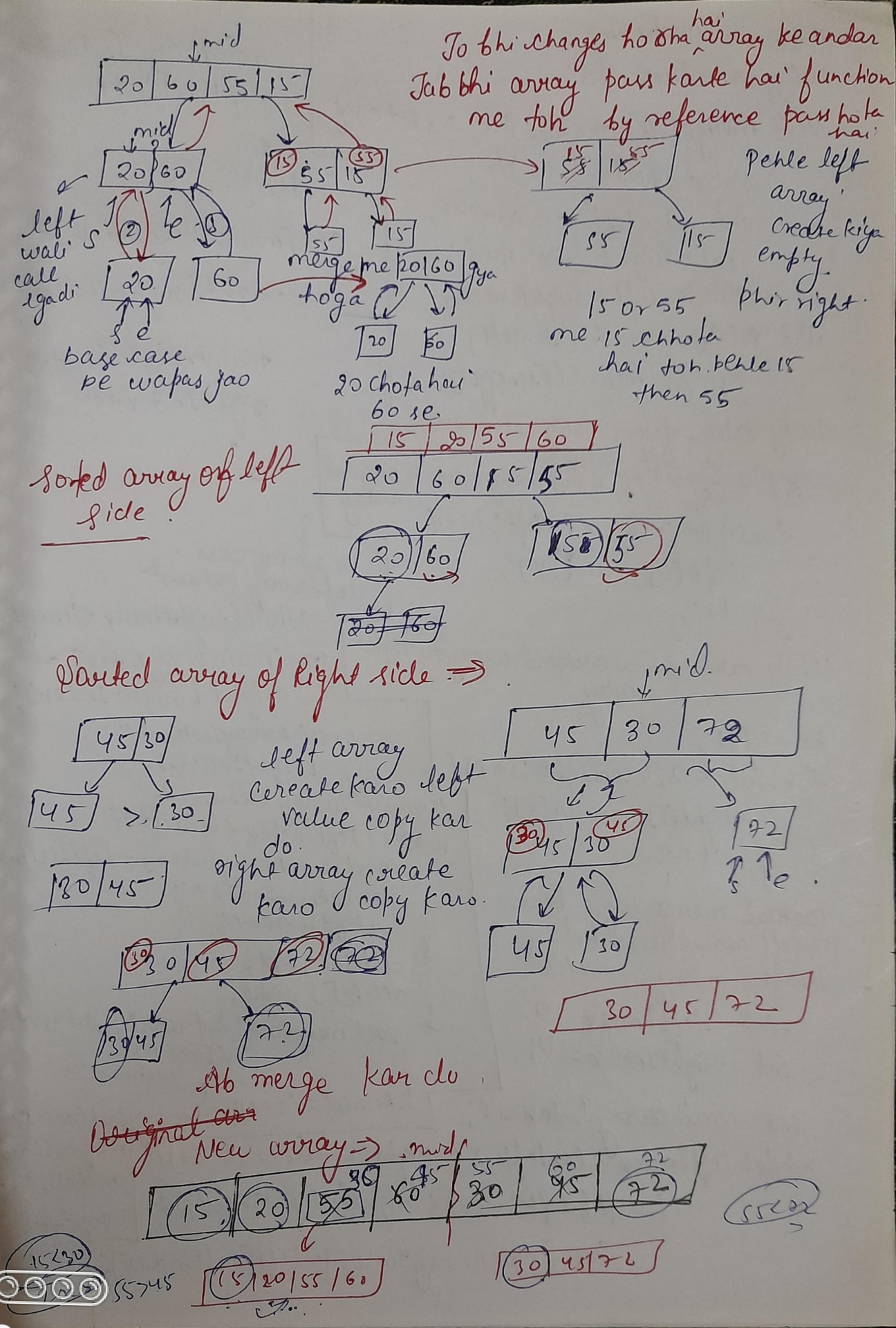
Heap me jab bhi memory allocate karwa rhe hai toh manually usko deallocate bhi karo.

deallocate `delete [ ] arr;`

```
mergeSort(arr, s, e){
    // BC.
    → Break into L & R
    → Recursion for L & R
    → merge L & R
}
```

```
merge(arr, s, e, mid){
    → [create left & right array]
    → [copy values from actual array into left & right array]
    → [actual merge 2 sorted array (using 2 pointer)]
}
```




mid
20 60 55 15
mid
20 60
15 55 55 15
left wali call jagah
s e
20
60
s e
base case pe wapas jao
55
15
merge me 20 60 jaga hoga
20
60
20 chota hai 60 se.
Jo bhi changes hota hai array ke andar Jab bhi array pass karte hai function me toh by reference pass hota hai
15 55 15 55
55
15
Pehle left array create kiya empty. Phir right.
15 or 55 me 15 chhota hai toh pehle 15 then 55
Sorted array of left side
15 20 55 60
20 60 15 55
20 60
15 55
20 60
Sorted array of Right side ⇒
45 30
45 > 30
30 45
left array create karo left value copy kar do.
right array create karo copy karo.
mid
45 30 72
30 45
45
30
72
30 45 72
30 45 72
30 45
72
Ab merge kar do.
Original array
New array →
mid
15 20 55 60 45 55 30 60 45 72 72
15 20 55 60
30 45 72
15<20
55>45
55<72

#include <iostream>
using namespace std;
void merge(int arr[], int s, int e){
int mid = (s+e)/2;
int lenLeft = mid-s+1;
int lenRight = e-mid;
// create left & right array
int *left = new int[lenLeft];
int *right = new int[lenRight];
//copy values from original array to left array
k -> starting index of left array
int k = s;
for(int i=0; i<lenLeft; i++){
left[i] = arr[k];
k++;
}
// copy values from original array to right array
k = mid+1;
for(int i=0; i<lenRight; i++){
right[i] = arr[k];
k++;
}
//actual merge logic here
// left array is sorted
// right array is sorted.
int leftIndex = 0;
int rightIndex = 0;
int mainArrayIndex = s;
while(leftIndex < lenLeft && rightIndex < lenRight){
if(left[leftIndex] < right[rightIndex]){
arr[mainArrayIndex] = left[leftIndex];
mainArrayIndex++;
leftIndex++;
}
else{
arr[mainArrayIndex] = right[rightIndex];
mainArrayIndex++;
rightIndex++;
}
}
// 2 more case.
// left array exhaust
while(rightIndex < lenRight){
arr[mainArrayIndex] = right[rightIndex];
mainArrayIndex++;
rightIndex++;
}
// right array exhaust
while(leftIndex < lenLeft){
arr[mainArrayIndex] = left[leftIndex];
mainArrayIndex++;
leftIndex++;
}
delete[] left;
delete[] right;
}
void mergeSort(int arr[], int s, int e){
if(s >= e){
return;
}
int mid = (s+e)/2;
mergeSort(arr, s, mid);
mergeSort(arr, mid+1, e);
merge(arr, s, e);
}

HW: [Inversion Count
In Place Merge Sort]

Time Complexity ⇒ O(nlogn)

MS( ){
IIBC
MS(half)
MS(half)
merge( )
}

$T(n) = k_1 + T(n/2) + T(n/2) + n*k$ (BC, Left, Right, merge)

$T(n) = 2k_1 + 2T(n/2) + n*k$

$2*T(n/2) = 2k_1 + 2\cdot 2T(n/4) + \frac{n*k}{2} \times 2$

$4*T(n/4) = 4k_1 + 8\cdot T(n/8) + \frac{n}{4}*k*4$

$8*T(n/8) = 8k_1 + 16\cdot T(n/16) + \frac{n}{8}*k*8$

$2^{a-1}*T(1) = 2^{a-1}k_1$ (n*k)

$T(n) = k_1(1+2+4+\ldots 2^{a-1}) + (a-1)*n*k$

$q(n) = k(\underbrace{1+2+4+\ldots 2^{a-1}}_{n}) + (a-1)*n*k.$

$= nk_1 + a*n*k$

$= n(k_1 + a*k)$

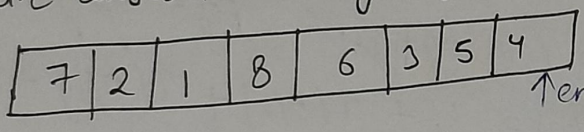
$= n \times a \times k.$

$= nxa = n\log n.$

$T(n) = n\log n$

Quick Sort

Given an unsorted array. you need to sort it



Take a pivot

→ To place pivot such that the elt to the right of pivot > a[pivot].

→ element left of pivot ele < a[pivot].

↳ New way to do partioning →

pivot = end

① pivot

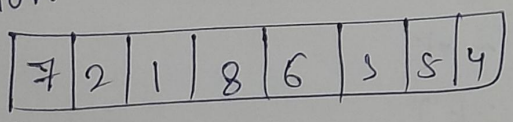
② i = start - 1;

j = start

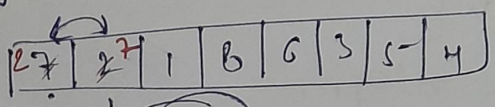
a[pivot] = 4

```
while(j < pivot){
    if(a[j] < a[pivot]){
        i++;
        swap(a[i], a[j]);
    }
    j++;
}
i++;
swap(a[i], a[pivot]);
```

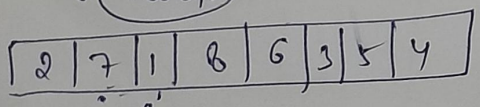
Iteration 3



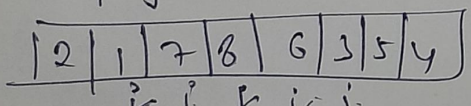
Iteration



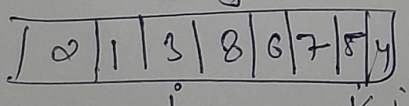
Iteration



Iteration

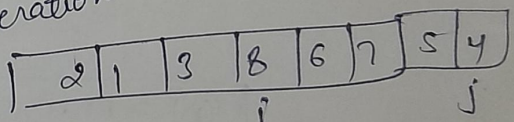


Iteration

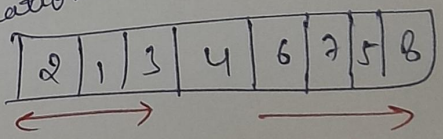


j = upperbound

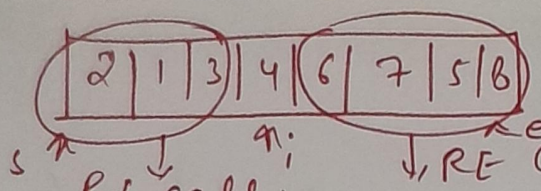
Iteration

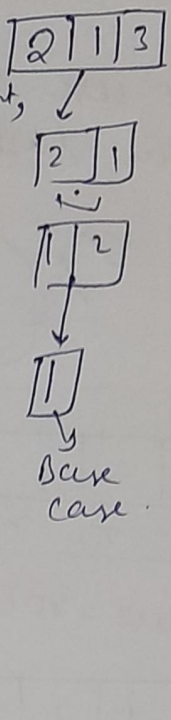


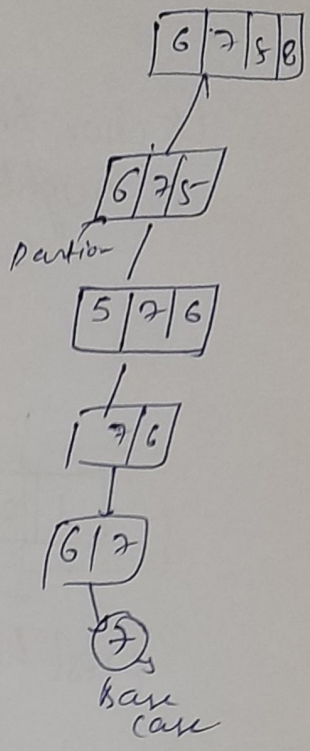
Iteration



4. sekan ⑨ 4 se jada








```
void quicksort (int arr[], int start,
                int end){
if (start >= end) return;
int pivot = end;
int i = start-1;
int j = start;
while (j < pivot){
 if (a[j] < a[pivot]){
        ++i;
     swap(a[i], a[j]);
 } ++j;
}
 ++i;  // i is the right position for the pivot element.
 swap (a[i], a[pivot])
 quicksort (a, start, i-1);
 quicksort (a, i+1, end);
}
```