Quick Sort

Quicksort कार द्वारा आविष्कार एक बहुत ही कुशल छँटाई कलन विधि है Hoare. यह दो चरणों है:

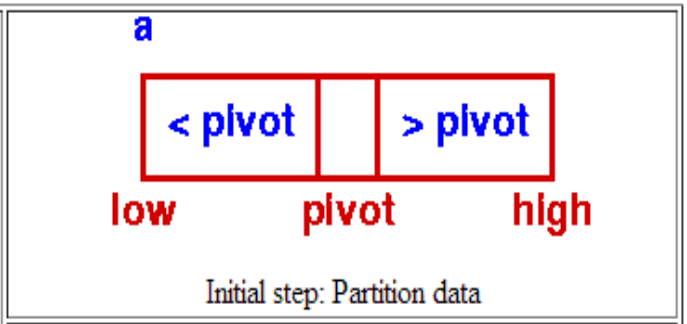
• विभाजन चरण और

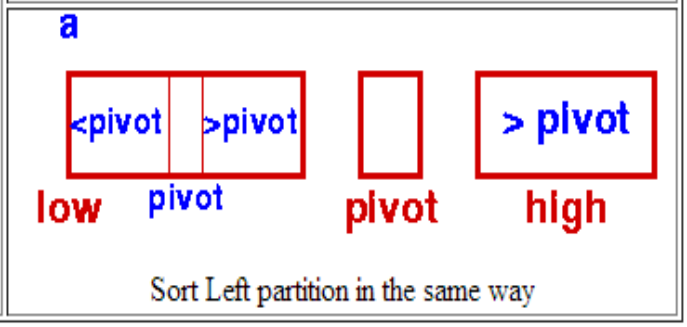
सॉर्ट चरण •.

जैसा कि हम देखेंगे, काम के अधिकांश विभाजन चरण में किया जाता है - यह जहां काम को विभाजित करने के लिए बाहर काम करता है. सॉर्ट चरण बस विभाजन चरण में उत्पन्न कर रहे हैं कि दो छोटी समस्याओं सॉर्ट करता है.

इस quicksort समस्याओं को सुलझाने के लिए विभाजन और जीत की रणनीति का एक अच्छा उदाहरण है. (आप पहले से ही द्विआधारी खोज की प्रक्रिया में इस दृष्टिकोण का एक उदाहरण देखा है.) Quicksort में, हम दो विभाजन में हल किया जा करने के लिए आइटम की सरणी के विभाजन और उसके बाद दो विभाजन सॉर्ट करने के लिए बारी बारी से quicksort प्रक्रिया कॉल, यानी हम विभाजित दो छोटे में समस्या और छोटे लोगों को हल करके जीत. Quicksort दिनचर्या का जीतना हिस्सा इस तरह दिखता है:

```
quicksort( void *a, int low, int high )
  {
  int pivot;
  /* Termination condition! */
  if ( high > low )
    {
    pivot = partition( a, low, high );
    quicksort( a, low, pivot-1 );
    quicksort( a, pivot+1, high );
    }
  }
```





रणनीति प्रभावी होने के लिए, विभाजन चरण धुरी एक हिस्सा (निचले भाग) में सभी मदों से अधिक और अन्य (ऊपरी) भाग में उन सभी की तुलना में कम है कि यह सुनिश्चित करना चाहिए. ऐसा करने के लिए, हम एक धुरी तत्व का चयन और निचले हिस्से में सभी आइटम धुरी से भी कम है और यह अधिक से अधिक ऊपरी भाग में उन सभी कर रहे हैं कि व्यवस्था. धुरी तत्व के लिए किसी भी विकल्प क्या करेंगे इतना है कि सबसे सामान्य स्थिति में, हम हल किया जाना वस्तुओं के बारे में कुछ भी पता नहीं है - पहला तत्व कई

मदों धुरी के रूप में वही कर रहे हैं one.If एक सुविधाजनक है, इन वस्तुओं एक तिहाई (मध्य) विभाजन में धुरी के साथ समूहीकृत या निचले भाग में छोड़ा जा सकता है: "कम से कम या बराबर" के लिए ऊपर वर्णन में परिवर्तन "से कम".

CODES(C)

```
#include<stdio.h>

void swap (int a[], int left, int right)
{
 int temp;
 temp=a[left];
 a[left]=a[right];
 a[right]=temp;
}//end swap

void quicksort( int a[], int low, int high )
{
 int pivot;
 // Termination condition!
 if ( high > low )
 {
  pivot = partition( a, low, high );
  quicksort( a, low, pivot-1 );
  quicksort( a, pivot+1, high );
 }
} //end quicksort

int partition( int a[], int low, int high )
{
 int left, right;
 int pivot_item;
 int pivot = left = low;
 pivot_item = a[low];
 right = high;
 while ( left < right )
 {
  // Move left while item < pivot
  while( a[left] <= pivot_item )
   left++;
  // Move right while item > pivot
  while( a[right] > pivot_item )
   right--;
  if ( left < right )
   swap(a,left,right);
 }
 // right is final position for the pivot
 a[low] = a[right];
 a[right] = pivot_item;
 return right;
}//end partition

// void quicksort(int a[], int, int);
void printarray(int a[], int);

int main()
{
 int a[50], i, n;
 printf("\nEnter no. of elements: ");
```

```
 scanf("%d", &n);
 printf("\nEnter the elements: \n");
 for (i=0; i<n; i++)
  scanf ("%d", &a[i]);
 printf("\nUnsorted elements: \n");
 printarray(a,n);
 quicksort(a,0,n-1);
 printf("\nSorted elements: \n");
 printarray(a,n);

}//end main


void printarray(int a[], int n)
{
 int i;
 for (i=0; i<n; i++)
  printf(" %d ", a[i]);
 printf("\n");
}//end printarray
```

CODES(JAVA)

```
import java.io.*;
import java.util.*;

public class QuickSort
{
  public static void swap (int A[], int x, int y)
  {
    int temp = A[x];
    A[x] = A[y];
    A[y] = temp;
  }

  // Reorganizes the given list so all elements less than the first are
  // before it and all greater elements are after it.
  public static int partition(int A[], int f, int l)
  {
    int pivot = A[f];
    while (f < l)
    {
      if (A[f] == pivot || A[l] == pivot)
      {
        System.out.println("Only distinct integers allowed - C321");
        System.out.println("students should ignore this if statement");
        System.out.exit(0);
      }
      while (A[f] < pivot) f++;
      while (A[l] > pivot) l--;
      swap (A, f, l);
    }
    return f;
  }

  public static void Quicksort(int A[], int f, int l)
  {
    if (f >= l) return;
    int pivot_index = partition(A, f, l);
    Quicksort(A, f, pivot_index);
    Quicksort(A, pivot_index+1, l);
  }

  // Usage: java QuickSort [integer] ...
  // All integers must be distinct
  public static void main(String argv[])
  {
    int A[] = new int[argv.length];
    for (int i=0 ; i < argv.length ; i++)
      A[i] = Integer.parseInt(argv[i]);
```

```
    Quicksort(A, 0, argv.length-1);

    for (int i=0 ; i < argv.length ; i++) System.out.print(A[i] + " ");
    System.out.println();
  }
}
```