Unidad 2

Fundamentos de Programación Competitiva

Logro de sesión

Al finalizar la sesión, el estudiante comprenderá algoritmos de ordenamientos avanzados

Semana 5

Estructuras de datos

Contenido:

- Algoritmos de ordenamiento rápido
- Ordenamiento por mezcla
- Ordenamiento por montículos
- Ordenamiento por cuentas

Ordenamiento Rápido

QUICKSORT

- Inventado por CAR Hoare, 1960

- El ordenamiento rápido (Quicksort en ingles) es un algoritmo basado en la técnica divide y vencerás, que permite ordenar n elementos en un tiempo proporcional a O (n log (n)).

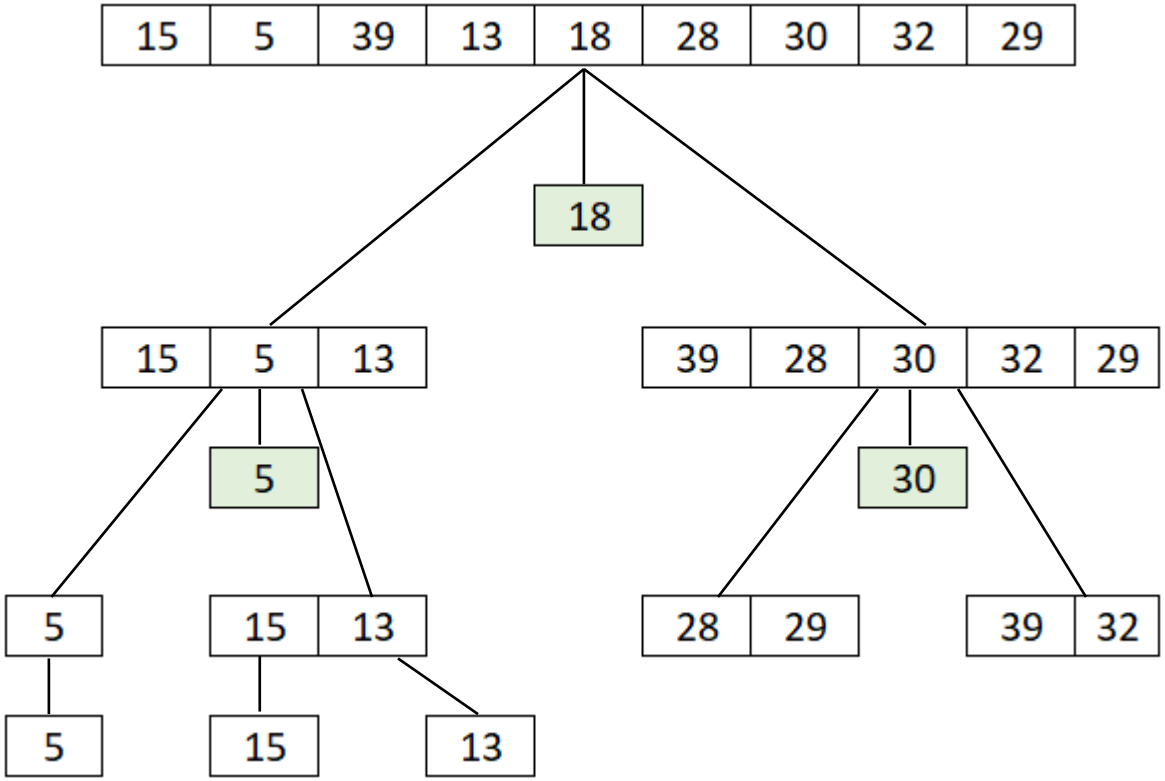
Ordenamiento Rápido

QUICKSORT

- Elegir un elemento del vector a ordenar, al que llamaremos pivote.
- Mover todos los elementos menores que el pivote a un lado y los mayores al otro lado.
- El vector queda separado en 2 sub vectores una con los elementos a la izquierda del pivote y otra con los de la derecha.
- Repetir este proceso recursivamente en las sub listas hasta que estas tengan un solo elemento.

Ordenamiento Rápido

QUICKSORT



Ordenamiento Rápido

QUICKSORT

```
1   #include <iostream>
2   using namespace std;
3
4   int particion(int *A, int p, int r) {
5       int x = A[r]; //el pivote
6       int i = p - 1; //índice de los menores
7       for (int j = p; j < r; j++) {
8           if (A[j] <= x) {
9               i++;
10              swap(A[i],A[j]);
11          }
12      }
13      swap(A[i + 1], A[r]);
14      return i + 1;
15  }
16
17  void quicksort(int *A, int p, int r) {
18      int q; //para almacenar el índice del pivote
19
20      if (p < r) {
21          q = particion(A, p, r); //devuelve el índice del pivote
22          quicksort(A, p, q - 1);
23          quicksort(A, q + 1, r);
24      }
25  }
```

Ordenamiento por Mezcla

MERGESORT

- Fue desarrollado en 1945 por John Von Neumann.

- El algoritmo de ordenamiento por mezcla (Merge Sort) es un algoritmo de ordenación externo estable basado en la tecnica divide y vencerás. Su complejidad es O(nlogn).

Ordenamiento por Mezcla

MERGESORT

- Si la longitud del vector es 1 o 0, entonces ya esta ordenado, en otro caso:
- Dividir el vector desordenado en dos sub vectores de aproximadamente la mitad de tamaño.
- Ordenar cada sub vector recursivamente aplicando el ordenamiento por mezcla.
- Mezclar los dos sub vectores en un solo sub vector ordenado.

Ordenamiento por Mezcla

MERGESORT

```
1  void mergeSort(int *Vector, int n) {
2      if (n > 1) {
3          int mitad = n / 2;
4          int *Vector1 = new int[mitad];
5          int *Vector2 = new int[n-mitad];
6
7          for (int i = 0;i < mitad;i++) {
8              Vector1[i] = Vector[i];
9          }
10         for (int i = mitad;i < n;i++) {
11             Vector2[i-mitad] = Vector[i];
12         }
13         mergeSort(Vector1, mitad);
14         mergeSort(Vector2, n - mitad);
15         Merge(Vector1, Vector2, Vector, n);
16     }
17 }
```

Continúa…

Ordenamiento por Mezcla

MERGESORT

```
19  void Merge(int* Vector1, int* Vector2, int* Vector, int n){
20      int i = 0, j = 0, k = 0;
21      int mitad = n / 2;
22      while (i < mitad && j < n - mitad) {
23          if (Vector1[i] < Vector2[j]) {
24              Vector[k] = Vector1[i];
25              i++; k++;
26          }
27          else {
28              Vector[k] = Vector2[j];
29              j++; k++;
30          }
31      }
32      while (i < mitad) {
33          Vector[k] = Vector1[i];
34          i++; k++;
35      }
36      while (j < n - mitad) {
37          Vector[k] = Vector2[j];
38          j++; k++;
39      }
40  }
```

Ordenamiento por Montículos

HEAPSORT

- Este algoritmo consiste en almacenar todos los elementos del vector a ordenar en un montículo (heap),
- y luego extraer el nodo que queda como nodo raíz del montículo (cima) en sucesivas iteraciones obteniendo el conjunto ordenado.

- Es un algoritmo de ordenación no recursivo, con complejidad O(nlogn).

Ordenamiento por Montículos

HEAPSORT

```
1   int parent(int i) {
2       return (i - 1) / 2;
3   }
4   int left(int i) {
5       return 2 * i + 1;
6   }
7   int rigth(int i) {
8       return 2 * i + 2;
9   }
10
11  void maxHeapify(int A[], int n, int i) {
12      int l = left(i);
13      int r = rigth(i);
14      int largest(0);
15      if (l <= (n - 1) && A[l] > A[i]) {
16          largest = l;
17      }else
18          largest = i;
19      if (r <= (n - 1) && A[r] > A[largest]) {
20          largest = r;
21      }
22      if (largest != i) {
23          swap(A[i], A[largest]);
24          maxHeapify(A, n, largest);
25      }
26  }
```

Ordenamiento por Montículos

HEAPSORT

```
28  void buildMaxHeap(int A[], int n) {
29      for (int i = n / 2 - 1;i >= 0;--i) {
30          maxHeapify(A, n, i);
31      }
32  }
33
34  //aplicación de ordenamiento
35  void heapsort(int A[], int n) {
36      buildMaxHeap(A, n);
37      for (int i = n - 1;i > 0;--i) {
38          swap(A[0], A[i]);
39          maxHeapify(A, --n, 0);
40      }
41  }
```

Ordenamiento por Cuentas

COUNTING SORT

```
40 void counting_sort(int A[], int Aux[], int sortedA[], int N) {
41
42     // Buscamos maximo valor de A[]
43     int K = 0;
44     for(int i=0; i<N; i++)   K = max(K, A[i]);
45
46     // Inicializamos Aux[] con 0
47     for(int i=0 ; i<=K; i++)   Aux[i] = 0;
48
49     // Guardamos las frecuencias de A[],
50     // mapeando los valores en Aux[]
51     for(int i=0; i<N; i++)  Aux[A[i]]++;
52
53     int j = 0;
54     for(int i=0; i<=K; i++) {
55         int tmp = Aux[i];
56         // Añadimos i al arreglo ordenado el numero de veces que aparecio
57         while(tmp--) {
58             //cout << Aux[i] << endl;
59             sortedA[j] = i;
60             j++;
61         }
62     }
63 }
64
```

Muchas Gracias!!!