***Luis F. Muñoz****Monitoria Estructura de Datos*

**Busqueda**

**//Busqueda lineal**

int buscar(int arr[], int n, int x)

{

int i;

for (i = 0; i < n; i++)

if (arr[i] == x)

return i;

return -1;

}

//Fin de Busqueda lineal

**//Busqueda Binaria**

int busquedaBinaria(int arr[], int l, int r, int x)

{

if (r >= l) {

int mid = l + (r - l) / 2;

if (arr[mid] == x)

return mid;

if (arr[mid] > x)

return busquedaBinaria(arr, l, mid - 1, x);

return busquedaBinaria(arr, mid + 1, r, x);

}

return -1;

}

**//Fin de Busqueda Binaria**

**//fin de Busqueda**

**ORDENAMIENTO**

**Función auxiliar para intercambiar elementos:**

void swap(int \*xp, int \*yp)

{

    int temp = \*xp;

    \*xp = \*yp;

    \*yp = temp;

}

**Ordenamiento Burbuja**

void ordenamientoBurbuja(int arr[], int n)

{

   int i, j;

   for (i = 0; i < n-1; i++)

       for (j = 0; j < n-i-1; j++)

           if (arr[j] > arr[j+1])

              swap(&arr[j], &arr[j+1]);

}

**Fin Ordenamiento Burbuja**

**Ordenamiento de Selección**

void OrdSeleccion(int arr[], int n)

{

    int i, j, min\_idx;

        for (i = 0; i < n-1; i++)

    {

        min\_idx = i;

        for (j = i+1; j < n; j++)

          if (arr[j] < arr[min\_idx])

            min\_idx = j;

        swap(&arr[min\_idx], &arr[i]);

    }

}

**Fin de Ordenamiento de Selección**

**Ordenamiento de Quicksort**

int dividir (int arr[], int izq, int der)

{

    int pivot = arr[der];

    int i = (low - 1);

    for (int j = izq; j <= der- 1; j++)

    {

        if (arr[j] <= pivot)

        {

            i++;

            swap(&arr[i], &arr[j]);

        }

    }

    swap(&arr[i + 1], &arr[der]);

    return (i + 1);

}

void quickSort(int arr[], int izq, int der)

{

    if (izq < der)

    {

        int pi = dividir (arr, izq, der);

        quickSort(arr, izq, pi - 1);

        quickSort(arr, pi + 1, der);

    }

}

**Fin de Ordenamiento de Quicksort**

**Ordenamiento MergeSort**

void merge(int arr[], int l, int m, int r)

{

    int i, j, k;

    int n1 = m - l + 1;

    int n2 =  r - m;

    int L[n1], R[n2];

      for (i = 0; i < n1; i++)

        L[i] = arr[l + i];

    for (j = 0; j < n2; j++)

        R[j] = arr[m + 1+ j];

    i = 0;

    j = 0;

    k = l;

    while (i < n1 && j < n2)

    {

        if (L[i] <= R[j])

        {

            arr[k] = L[i];

            i++;

        }

        else

        {

            arr[k] = R[j];

            j++;

        }

        k++;

    }

    while (i < n1)

    {

        arr[k] = L[i];

        i++;

        k++;

    }

    while (j < n2)

    {

        arr[k] = R[j];

        j++;

        k++;

    }

}

void mergeSort(int arr[], int l, int r)

{

    if (l < r)

    {

        int m = l+(r-l)/2;

        mergeSort(arr, l, m);

        mergeSort(arr, m+1, r);

        merge(arr, l, m, r);

    }   
}

**Fin de Ordenamiento de MergeSort**

**FIN ORDENAMIENTO**

**PLANTILLAS   
Saca el máximo de dos números.**

template <typename T>

T Max (T a, T b) {

return a < b ? b:a;

}

**FIN DE PLANTILLAS**

**BACKTRACKING**

**Revisar el código de las NReinas, esta en mi cuenta de GitHub:** [**https://github.com/JarvisFullStack/DataStructureUcne/blob/master/Backtracking/ExampleProblems/NQueen.cpp**](https://github.com/JarvisFullStack/DataStructureUcne/blob/master/Backtracking/ExampleProblems/NQueen.cpp)

**FIN BACKTRACKING**

**ANALISIS DE ALGORITMOS**

1. **Cualquier llamada a una función u operación realizada tiene complejidad de   
   O(1).  
   Ej: int a = 1+2;   
   sumar(a,2);**
2. **Un ciclo de la siguiente manera  
   for(int i =0;i<10;i++) tiene una complejidad de  
    O(n)**
3. **Dos ciclos anidados de la siguiente manera**

for (int i = 1; i <=n; i += c) {

for (int j = 1; j <=n; j += c) {

}

}  
**Tiene la complejidad de O(nc) en este caso O(n2).**

1. **Un ciclo realizando una operación de multiplicación o división como los siguientes tienen complejidad de O(Log n)**for (int i = 1; i <=n; i \*= c) { }

for (int i = n; i > 0; i /= c) { }

1. **Un ciclo que incremente o decremente de manera exponencial tiene la complejidad de O(Log Log n)**for (int i = 2; i <=n; i = pow(i, c)) { }

for (int i = n; i > 0; i = fun(i)) { }

**FIN DE ANALISIS DE ALGORITMOS**