GUIA DE ESTUDIO PARA DAA

MERGE SORT

```c
#define N 100

void merge(int left, int centro, int right, int a[]){
  int temp[N];
  int x,y,i;
  x=i=left; //x representa el indice temporal del lado izquierdo
  y = centro + 1;// y representa el indice temporal del lado derecho

  //aqui alguno de los dos lados se va a quedar vacío
  while(x<=centro && y<= right){
    if(a[x] <= a[y])
      temp[i++] = a[x++];
    else
      temp[i++] = a[y++];
  }

  //luego, vaciar el lado que se haya quedado con elementos
  while(x<=centro)
    temp[i++] = a[x++];

  while(y<=right)
    temp[i++] = a[y++];

  //copiar el merge al arreglo original desde los índices mandados
  for(i = left; i<= right; i++)
    a[i] = temp[i]; //en el temp se almacenó el merge en las mismas posiciones del arreglo original
}

void mergeSort(int left, int right, int a[]){
  if(left<right){
    int centro = (left + right)/2;
    mergeSort(left, centro, a);
    mergeSort(centro+1, right, a);
    merge(left, centro, right, a);
  }
}
int main() {
  int a[] = {4,3,2,1};
  mergeSort(0, 3, a);

  return 0;
}
```

QUICK SORT LOMUTO

```c
#include<stdio.h>

void swap(int *a, int *b){
  int temp = *a;
  *a = *b;
  *b = temp;
  }

int partition(int arr[],int low,int high)
{
  int pivot=arr[high];
  int i=(low-1);

  for(int j=low;j<=high;j++)
  {
    if(arr[j]<pivot)
    {
      i++;
      swap(&arr[i],&arr[j]);
    }
  }
  swap(&arr[i+1],&arr[high]);
  return (i+1);
}

void quickSort(int arr[],int low,int high)
{
  if(low<high)
  {
    int pi=partition(arr,low,high);
    quickSort(arr,low,pi-1);
    quickSort(arr,pi+1,high);
  }
}


int main(){
  int a[] = {10, 80, 30, 90, 40};
  int n = (int) sizeof(a) / sizeof(int);

  quickSort(a, 0, n-1);

  return 0;
}
```

GUIA DE ESTUDIO PARA DAA

QUICK SORT HUARE

```c
#include<stdio.h>
#include<time.h>
#include<stdlib.h>

void swap(int *a, int *b){
    int temp = *a;
    *a = *b;
    *b = temp;
}

int partition(int a[], int left, int right){
    int i = left, j = right-1;
    int pivote = a[right]; //último

    while(i <= j){
        while(a[i] < pivote)
            i++;

        while(j>=left && a[j] > pivote)
            j--;

        if(i <= j){
            swap(&a[i], &a[j]);
            i++;
            j--;
        }
    }

    swap(&a[j + 1], &a[right]);
    return j+1;
}

void quicksort(int a[], int left, int right){
    if(left>=right)
        return;

    int p = partition(a, left, right);
    quicksort(a, left, p - 1);
    quicksort(a, p + 1, right);

}



int main() {
    int a[10] = {14, 25, 44, 43, 25, 41, 22, 12, 7, 16};
    int n = 10;
    int i, j;

    quicksort(a, 0, n-1);

    return 0;
}
```

COUNTING SORT

```c
#include <stdio.h>
#include <stdlib.h>

// función para encontrar el valor máximo en el arreglo
int encontrarMaximo(int arr[], int n) {
    int max = arr[0];
    for (int i = 1; i < n; i++) {
        if (arr[i] > max) {
            max = arr[i];
        }
    }
    return max;
}

// función de counting sort
void countingSort(int arr[], int n) {
    int i;
    int max = encontrarMaximo(arr, n);

    // crear el arreglo de conteo
    int *conteo = (int*)calloc((max+1), sizeof(int));

    // contar la ocurrencia de cada elemento
    for (i = 0; i < n; i++) {
        conteo[arr[i]]++;
    }

    // reconstruir el arreglo original ordenado
    int idx = 0;
    for (i = 0; i <= max; i++) {
        while (conteo[i] > 0) {
            arr[idx++] = i;
            conteo[i]--;
        }
    }

    free(conteo); // liberar la memoria del arreglo de
conteo
}

void imprimirArreglo(int arr[], int n) {
    for (int i = 0; i < n; i++) {
        printf("%d ", arr[i]);
    }
    printf("\n");
}

int main() {
    int arr[] = {4, 2, 2, 8, 3, 3, 1};
    int n = sizeof(arr) / sizeof(arr[0]);

    printf("Arreglo original:\n");
    imprimirArreglo(arr, n);

    countingSort(arr, n);

    printf("Arreglo ordenado:\n");
    imprimirArreglo(arr, n);

    return 0;
}
```

BUSQUEDA BINARIA

```
int binarySearch(int a[], int l, int r, int elem){
  if(l>r)
    return -1; //recuerden que es un índice
  int centro = (l+r)/2;
  //Nuestro caso base (:
  if(a[centro] == elem)
    return centro;//nuestro índice

  if(a[centro]>elem)
    return binarySearch(a, l, centro-1, elem);
  return binarySearch(a, centro+1, r, elem);
}

int main() {
  int a[] = {10,25, 35, 50, 100, 120, 200, 400,900, 1000};

  int elementoBuscado = binarySort(a, 0, 9, 78);
  return 0;
}
```

GUIA DE ESTUDIO PARA DAA

HEAP SORT

```c
#include<stdio.h>

typedef struct heap{
  int *a;
  int n;
}Heap;

//inicialización del montículo
void init(Heap * h, int space){
  h->a = (int *) malloc(sizeof(int)*space);
  h->n = 0;
}

void swap(int *a, int *b){
  int temp = *a;
  *a = *b;
  *b = temp;
}


void push(Heap *h, int dato){
  //primer paso: insertar al final
  int temp, i = h->n;
  h->a[i] = dato;
  h->n++;

  //flotar
  for(;i > 0; i=(i-1)/2){
    //si el hijo es menor que el padre, debe flotar
    if(h->a[i] < h->a[(i-1)/2]){
      swap(&h->a[i],&h->a[(i-1)/2]);
    }
  }

}

int pop(Heap *h){
  //guardar el dato a retornar
  int i=0, temp = h->a[0];
  //hacer que el ultimo elemento sea el primero
  h->a[0] = h->a[(h->n)-1];
  h->n--;

  //hundirlo
  int hMenor;
  while(i < h->n/2){
    if(h->a[2*i+1] <  h->a[2*i+2])
      hMenor = 2*i+1;
    else
      hMenor = 2*i+2;

    if(h->a[i] > h->a[hMenor]){
      swap(&h->a[i], &h->a[hMenor]);
      i = hMenor; //ahora el padre es el hijo
    }else
      break;
  }


  return temp;

}


int main() {
  int a[] = {12,6,3,7,1,8};
  int n = (int) sizeof(a)/sizeof(int);

  Heap h;
  init(&h, n);
  for(int i=0; i<n; i++)
    push(&h, a[i]);

  for(int i=0; i<n; i++)
    a[i] = pop(&h);

  return 0;
}
```

GUIA DE ESTUDIO PARA DAA

RADIX SORT

```c
#include <stdio.h>

// Función para obtener el número máximo
int obtenerMax(int arr[], int n) {
    int max = arr[0];
    for (int i = 1; i < n; i++)
        if (arr[i] > max)
            max = arr[i];
    return max;
}

// Counting Sort para un dígito específico (exp = 1, 10, 100...)
void countingSort(int arr[], int n, int exp) {
    int output[n];
    int count[10] = {0};

    // Contar ocurrencias de dígitos
    for (int i = 0; i < n; i++)
        count[(arr[i] / exp) % 10]++;

    // Convertir count[i] en posiciones
    for (int i = 1; i < 10; i++)
        count[i] += count[i - 1];

    // Construir el arreglo de salida
    for (int i = n - 1; i >= 0; i--) {
        int digito = (arr[i] / exp) % 10;
        output[count[digito] - 1] = arr[i];
        count[digito]--;
    }

    // Copiar al arreglo original
    for (int i = 0; i < n; i++)
        arr[i] = output[i];
}

// Radix Sort principal
void radixSort(int arr[], int n) {
    int max = obtenerMax(arr, n);

    for (int exp = 1; max / exp > 0; exp *= 10)
        countingSort(arr, n, exp);
}

// Función para imprimir
void imprimir(int arr[], int n) {
    for (int i = 0; i < n; i++)
        printf("%d ", arr[i]);
    printf("\n");
}

// MAIN
int main() {
    int arr[] = {170, 45, 75, 90, 802, 24, 2, 66};
    int n = sizeof(arr)/sizeof(arr[0]);

    printf("Arreglo original:\n");
    imprimir(arr, n);

    radixSort(arr, n);

    printf("Arreglo ordenado:\n");
    imprimir(arr, n);

    return 0;
}
```

PROBLEMA DE LA MOCHILA PROG. DINAMICA (IA)

```c
#include <stdio.h>
#include <string.h>

// función máxima
int max(int a, int b) {
    return (a > b) ? a : b;
}

// función principal de mochila con memoización
int knapsack(int W, int peso[], int valor[], int n, int memo[][100]) {
    // caso base: no quedan objetos o no hay capacidad
    if (n == 0 || W == 0)
        return 0;

    // si ya está calculado, regresa el resultado almacenado
    if (memo[n][W] != -1)
        return memo[n][W];

    if (peso[n-1] > W) {
        // no se puede incluir el objeto n-1
        memo[n][W] = knapsack(W, peso, valor, n-1, memo);
    } else {
        // opción 1: no tomar el objeto n-1
        // opción 2: tomar el objeto n-1
        memo[n][W] = max(
            knapsack(W, peso, valor, n-1, memo),
            valor[n-1] + knapsack(W - peso[n-1], peso, valor, n-1, memo)
        );
    }
    return memo[n][W];
}

int main() {
    int peso[] = {1, 3, 4, 5};
    int valor[] = {1, 4, 5, 7};
    int n = sizeof(valor)/sizeof(valor[0]);
    int W = 7;

    // inicializar el arreglo de memoización a -1
    int memo[100][100];
    memset(memo, -1, sizeof(memo));

    int maxValor = knapsack(W, peso, valor, n, memo);

    printf("El valor máximo que se puede obtener es: %d\n", maxValor);

    return 0;
}
```

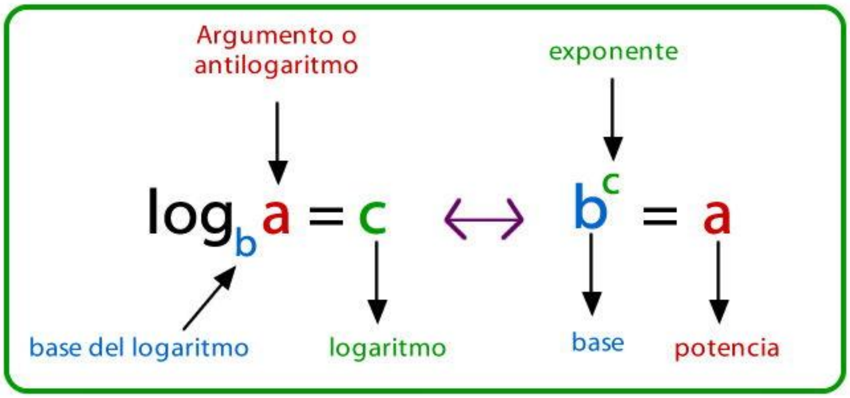
ORDENAMIENTO TOPOLOGICO (IA)

```c
#include <stdio.h>
#define N 4 // número de nodos

// Grafo como listas de adyacencia
int grafo[N][N] = { {0,1,1,0},
                    {0,0,0,1},
                    {0,0,0,1},
                    {0,0,0,0} };

int visitado[N];
int resultado[N];
int idx = N-1;

// DFS recursivo
void dfs(int nodo) {
    visitado[nodo] = 1;
    for(int i=0; i<N; i++) {
        if(grafo[nodo][i] && !visitado[i])
            dfs(i);
    }
    resultado[idx--] = nodo; // se coloca al final del orden
}

void ordenamiento_topologico() {
    for(int i=0; i<N; i++)
        visitado[i] = 0;

    for(int i=0; i<N; i++)
        if(!visitado[i])
            dfs(i);

    printf("Orden topologico: ");
    for(int i=0; i<N; i++)
        printf("%d ", resultado[i]);
    printf("\n");
}

int main() {
    ordenamiento_topologico();
    return 0;
}
```

ORDENAMIENTO TOPOLOGICO

```c
#include<stdlib.h>
#define NODOS 7

//matriz de adyacencia
int grafo[NODOS][NODOS]={};
int inDegree[NODOS] = {0};


void connectNodo(int origen, int destino){
    grafo[origen][destino] = 1;
}

void setInDegree(){
  for(int i =0; i < NODOS; i++)
    for(int j = 0; j < NODOS; j++)
      if( grafo[j][i] == 1)
        inDegree[i]++;
}

int findZeroInDegree(){
  int i = 0;
  for(; i < NODOS && inDegree[i]!=0; i++);
  if(i==NODOS)
    return -1;
  return i;
}

void deleteNodo(int nodo){
  inDegree[nodo] = -1;
  //al eliminar el nodo debemos quitar todas sus salidas
  //y por tanto eliminar las entradas de los nodos que dependian de este
  for(int i = 0; i<NODOS; i++){
    //buscar en la fila, los n nodos a los que estaba conectado
    if(grafo[nodo][i]==1)
      inDegree[i]--;
  }
}

int main() {
  int arr[NODOS];
  int nodo, i=0;

  //inserción de nodos al grafo
  connectNodo(0,1);
  connectNodo(0,3);
  connectNodo(0,2);
  connectNodo(1,3);
  connectNodo(1,4);
  connectNodo(2,5);
  connectNodo(3,5);
  connectNodo(3,2);
  connectNodo(3,6);
  connectNodo(4,6);
  connectNodo(4,3);
  connectNodo(6,5);
  //Poner el grado de entrada a cada nodito
  setInDegree();

  //El algoritmo en cuestión...
  do{
    nodo = findZeroInDegree();

    if(nodo!=-1){
      arr[i]=nodo;
      i++;

      deleteNodo(nodo); //decrementa los grados de entrada de los nodos dependientes
    }

  }while(nodo!=-1);

  return 0;
}
```

GUIA DE ESTUDIO PARA DAA



Relación	Complejidad	Ejemplo
T(n) = T(n/2) + O(1)	O(log n)	Búsqueda binaria
T(n) = T(n-1) + O(1)	O(n)	Búsqueda lineal, bucles for/while
T(n) = 2 T(n/2) + O(1)	O(n)	Recorrido de árbol binario (preorden, inorden, postorden)
T(n) = 2 T(n/2) + O(n)	O(n log n)	Merge Sort, Quick Sort
T(n) = T(n-1) + O(n)	O(n^2)	Ordenamiento por selección, ordenamiento burbuja
T(n) = 2 T(n-1) + O(1)	O(2^n)	Torres de Hanói, Backtracking total

La Sumatoria

Propiedades

a) $\sum_{i=m}^{n} k = (n - m + 1) \cdot k$ la sumatoria de una constante es igual al número de términos por la constante

b) $\sum_{i=m}^{n} k \cdot x_i = k \cdot \sum_{i=m}^{n} x_i \Rightarrow \sum_{i=m}^{n} k \cdot f(i) = k \cdot \sum_{i=m}^{n} f(i)$

c) $\sum_{i=m}^{n} (x_i + y_i) = \sum_{i=m}^{n} x_i + \sum_{i=m}^{n} y_i \Rightarrow \sum_{i=m}^{n} [f(i) + g(i)] = \sum_{i=m}^{n} f(i) + \sum_{i=m}^{n} g(i)$

Fórmula cerrada:

$$\sum_{i=1}^{n} i = \frac{n(n+1)}{2} \qquad \sum_{i=1}^{n} i^2 = \frac{n(n+1)(2n+1)}{6} \qquad \sum_{i=1}^{n} i^3 = \left[\frac{n(n+1)}{2}\right]^2 \qquad \sum_{i=1}^{n} a^i = \frac{a^{n+1}-1}{a-1}, a \neq 1$$

*Si el i empieza en 0, la fórmula cerrada se mantiene igual

2. 2.- Calcular T(n) para cada uno de los posibles casos, así como definir y **demostrar** con límites a qué órdenes de complejidad pertenece el siguiente programa.

```
void sort(int arr[], int n) {
    int i = 0;
    for (; i < n - 1; i++) {
        if (arr[i] > arr[i + 1]) {
            break;
        }
    }
    if(i == n-1) return;

    for (int i = 1; i < n; i++) {
        int clave = arr[i];
        int j = i - 1;

        while (j >= 0 && arr[j] > clave) {
            arr[j + 1] = arr[j];
            j--;
        }
        arr[j + 1] = clave;
    }
}
```

$$T_m(n) = t_1 + \sum_{i=0}^{n-2} t_2$$

$$T_m(n) = t_1 + (n-2-0+1)\,t_2$$

$$T_m(n) = \underset{C_2}{t_1} + (n-1)\underset{C_1}{t_2}$$

$$T_m(n) = C_1 n + C_2 \in \Omega(n)\,?$$

$$\lim_{n\to\infty} \frac{\cancel{\frac{n}{n}}^{1}}{\frac{\cancel{n}}{n}^{1} + \cancel{\frac{2}{n}}^{0}} = 1$$

$$\Omega(c_1 n + c_2) = \Omega(n)$$

$$c_1 n + c_2 \in \Omega(n)\ ✓$$

$$T_p(n) = t_1 + \sum_{i=1}^{n-1}\left(t_2 + \sum_{j=i-1}^{n-1} t_3\right)$$

$$T_p(n) = t_1 + \sum_{i=1}^{n-1} t_2 + \sum_{i=1}^{n-1}\sum_{j=i-1}^{n-1} t_3$$

$$T_p(n) = t_1 + (n-\cancel{1}-1+\cancel{1})\,t_2 + \sum_{i=1}^{n-1}(n-i-1-\cancel{1}+\cancel{1})\,t_3$$

$$t_1 + nt_2 - t_2 + (n\cancel{-1}-1\cancel{+1})(n-i-1)\,t_3$$

$$\underset{C_3}{t_1} + \underset{C_2}{nt_2} - t_2 + \underbrace{(n-1)(n-i-1)}_{n^2\ C_1}\,t_3$$

$$T_p(n) = C_1 n^2 + C_2 n + C_3$$

$$T_p(n) = C_1 n^2 + C_2 n + C_3 \in O(n^2)\,!$$

$C_1 = 2$
$C_2 = 3$
$C_3 = 4$

$$\lim_{n\to\infty} \frac{\cancel{\frac{n^2}{n^2}}^{1}}{\frac{2\cancel{n^2}}{\cancel{n^2}} + \cancel{\frac{3n}{n^2}} + \cancel{\frac{4}{n^2}}^{0}} = \frac{1}{2}$$

$$C_1 n^2 + C_2 n + C_3 \in O(n^2)\ ✓$$

$$T_{\frac{1}{2}}(n) = \frac{1}{n}\sum_{i=1}^{n} c_1 i^2 + c_2 i + c_3$$

$$T_{\frac{1}{2}}(n) = \frac{1}{n}\left(c_1\sum_{i=1}^{n} i^2 + c_2\sum_{i=1}^{n} i + \sum_{i=1}^{n} c_3\right)$$

$$T_{\frac{1}{2}}(n) = \cancel{\frac{1}{n}}\left(C_1\left(\frac{\cancel{n}(n+1)(2n+1)}{6}\right) + C_2\left(\frac{\cancel{n}(n+1)}{2}\right) + (\cancel{n}-1+1)C_3\right)$$

$$T_{\frac{1}{2}}(n) = C_1\frac{(n+1)(2n+1)}{6} + \frac{C_2(n+1)}{2} + C_3$$

$$T_{\frac{1}{2}}(n) = \frac{C_1}{6}\left(2n^2 + \overset{3n}{\widehat{n + 2n}} + 1\right) + \frac{C_2}{2}n + \frac{C_2}{2} + C_3$$

$$T_{\frac{1}{2}}(n) = \underset{K_1}{\frac{n^2}{2}} + \underset{K_2}{\left(\frac{C_1 + C_2}{2}\right)}n + \underset{K_3}{\left(\frac{C_1}{6} + \frac{C_2}{2} + C_3\right)}$$

$$T_{\frac{1}{2}}(n) = K_1 n^2 + K_2 n + K_3 \in \Theta(n^2)\,?$$

$K_1 = 1$ $K_2 = 2$ $K_3 = 3$

$$\lim_{n\to\infty} \frac{\cancel{\frac{n^2}{n^2}}^{1}}{\cancel{\frac{n^2}{n^2}}^{1} + \cancel{\frac{2n}{n^2}}^{0} + \cancel{\frac{3}{n^2}}^{0}} = 1$$

$$K_1 n^2 + K_2 n + K_3 \in \Theta(n^2)\ ✓$$

1. 1.- Calcular T(n) para cada uno de los posibles casos, así como definir y **demostrar** con límites a qué órdenes de complejidad pertenece el siguiente programa.

Hint: Recuerda que la **n** no se considera para evaluar los casos...

```c
int main() {
  int i,j, n;
  i = 2;
  scanf("%d", &n);

  for(; i<n && n!=10; i++){
    for(j=i; j<n; j++){
      printf("%d,%d", i,j);
      if(j<i)
        goto a;
      }
    }

    a{
    i=2;
      }

  return 0;
}
```

$$T_m(n) = t_1$$

$$t_1 \in \Omega(1)\ ?$$

$$\lim_{n\to\infty} \frac{1}{1} = 1$$

$$t_1 \in \Omega(1)\ ✓$$

$$T_p(n) = t_1 + \sum_{i=2}^{n-1}\sum_{j=i}^{n-1} t_2$$

$$T_p(n) = t_1 + \sum_{i=2}^{n-1}(n \cancel{-1} - i \cancel{+1})\,t_2$$

$$T_p(n) = t_1 + \sum_{i=2}^{n-1}(n-i)\,t_2$$

$$T_p(n) = t_1 + \sum_{i=3}^{n} n\,t_2 - \sum_{i=3}^{n} i\,t_2$$

$$T_p(n) = \overset{C_3}{t_1} + \overset{C_2}{(n-2)t_2}\,n - \overset{C_1}{\frac{n(n+1)}{2}}\,t_2$$

$$T_p(n) = \overset{1}{C_1}n^2 + \overset{2}{C_2}n + \overset{3}{C_3} \in O(n^2)\ ?$$

$$\lim_{n\to\infty} \frac{\frac{n^2}{n^2}}{\frac{n^2}{n^2} + \frac{2n}{n^2} + \frac{3}{n^2}} = 1$$

$$C_1n^2 + C_2n + C_3 \in O(n^2)\ ✓$$

$$T_{\frac{1}{2}}(n) = \frac{1}{n}\left(C_1\sum_{i=1}^{n} i^2 + C_2\sum_{i=1}^{n} i + \sum_{i=1}^{n} C_3\right)$$

$$T_{\frac{1}{2}}(n) = \frac{1}{\cancel{n}}\left(C_1\left(\frac{\cancel{n}(n+1)(2n+1)}{6}\right) + C_2\left(\frac{\cancel{n}(n+1)}{2}\right) + (\cancel{n}-1+1)C_3\right)$$

$$T_{\frac{1}{2}}(n) = \frac{1}{n}\left(C_1\frac{(n+1)(2n+1)}{6} + \frac{C_2(n+1)}{2} + C_3\right)$$

$$T_{\frac{1}{2}}(n) = \frac{C_1}{6}(2n^2 + n + 2n + 1) + \frac{C_2}{2}n + \frac{C_2}{2} + C_3$$

$$T_{\frac{1}{2}}(n) = \overset{K_1}{\frac{n^2}{2}} + \overset{K_2}{\left(\frac{C_1 + C_2}{2}\right)}n + \overset{K_3}{\left(\frac{C_1}{6} + \frac{C_2}{2} + C_3\right)}$$

$$T_{\frac{1}{2}}(n) = \overset{K_1}{\frac{n^2}{2}} + \overset{K_2}{\left(\frac{C_1 + C_2}{2}\right)}n + \overset{K_3}{\left(\frac{C_1}{6} + \frac{C_2}{2} + C_3\right)}$$

$$T_{\frac{1}{2}}(n) = K_1n^2 + K_2n + K_3 \in \Theta(n^2)\ ?$$

$K_1 = 3 \quad K_2 = 2 \quad K_3 = 1$

$$\lim_{n\to\infty} \frac{\frac{n^2}{n^2}}{\frac{K_1n^2}{n^2} + \frac{K_2n}{n^2} + \frac{K_3}{n^2}} = 1$$

$$K_1n^2 + K_2n + K_3 \in \Theta(n^2)$$