3. Realizar uma pesquisa dos seguintes algoritmos:

1. Algoritmos de ordenação:
2. Insertion Sort

#include <stdio.h>

#include <stdlib.h>

void insertionSort(int v[200], int n) {

int i, j, x;

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

x = v[i];

j = i - 1;

while(j >= 0 && v[j] > x) {

v[j+1] = v[j];

j--;

}

v[j+1] = x;

}

}

1. Selection Sort:

#include <stdio.h>

#include <stdlib.h>

void selectionSort(int v[200], int n) {

int i, j, aux, min;

for(i = 0; i < n-1; i++) {

min = i;

for(j = i+1; j < n; j++) {

if(v[j] < v[min]) {

min = j;

}

}

aux = v[i];

v[i] = v[min];

v[min] = aux; //troca

}

}

1. Bublle sort:

#include <stdio.h>

#include <stdlib.h>

void bubbleSort(int v[200], int n) {

int i, j, aux;

for(i = n-1; i > 0; i--) {

for(j = 0; j < i; j++) {

if(v[j] > v[j+1]) {

aux = v[j];

v[j] = v[j+1];

v[j+1] = aux; //troca

}

}

}

}

1. Quick sort:

void quickSort(int \*vetor, int inicio, int fim){

  int i, j, meio, aux;

   i = inicio;

   j = fim;

   meio = vetor[(inicio + fim) / 2];

   do{

      while(vetor[i] < meio)

         i++;

      while(vetor[j] > meio)

         j--;

      if(i <= j){

         aux = vetor[i];

         vetor[i] = vetor[j];

         vetor[j] = aux;

         i++;

         j--;

 }

   }while(i <= j); //do

   if(inicio < j)

      quickSort(vetor, inicio, j);

   if(i < fim)

      quickSort(vetor, i, fim);

}

1. Merge sort:

/\* descrição funcional:   
      1. Ordena valores numericos de um vetor atraves do algoritmo Merge Sort.  
        
   lista de parâmetros:  
      1. Dois vetores  
      2. Uma variavel contendo o tamanho do vetor  
              
   valores de retorno:  
      1. Não há valores definidos para o retorno dessa função.  
\*/

void mergeSort(int numbers[], int temp[], int array\_size)  
{  
  m\_sort(numbers, temp, 0, array\_size - 1);  
}  
  
/\* descrição funcional:   
      1. Auxilia na ordenação de valores numericos de um vetor atraves do   
      algoritmo Merge Sort.  
        
   lista de parâmetros:  
      1. Dois vetores  
      2. Duas variaveis inteiras  
              
   valores de retorno:  
      1. Não há valores definidos para o retorno dessa função.  
\*/  
void m\_sort(int numbers[], int temp[], int left, int right)  
{  
  int mid;  
  
  if (right > left)  
  {  
    mid = (right + left) / 2;  
    m\_sort(numbers, temp, left, mid);  
    m\_sort(numbers, temp, mid+1, right);  
  
    merge(numbers, temp, left, mid+1, right);  
  }  
}  
  
/\* descrição funcional:   
      1. Auxilia na ordenação de valores numericos de um vetor atraves do   
      algoritmo Merge Sort.  
        
   lista de parâmetros:  
      1. Dois vetores  
      2. Tres variaves inteiras.  
              
   valores de retorno:  
      1. Não há valores definidos para o retorno dessa função.  
\*/  
void merge(int numbers[], int temp[], int left, int mid, int right)  
{  
  int i, left\_end, num\_elements, tmp\_pos;  
  
  left\_end = mid - 1;  
  tmp\_pos = left;  
  num\_elements = right - left + 1;  
  
  while ((left <= left\_end) && (mid <= right))  
  {  
    if (numbers[left] <= numbers[mid])  
    {  
      temp[tmp\_pos] = numbers[left];  
      tmp\_pos = tmp\_pos + 1;  
      left = left +1;  
    }  
    else  
    {  
      temp[tmp\_pos] = numbers[mid];  
      tmp\_pos = tmp\_pos + 1;  
      mid = mid + 1;  
    }  
  }  
  
  while (left <= left\_end)  
  {  
    temp[tmp\_pos] = numbers[left];  
    left = left + 1;  
    tmp\_pos = tmp\_pos + 1;  
  }  
  while (mid <= right)  
  {  
    temp[tmp\_pos] = numbers[mid];  
    mid = mid + 1;  
    tmp\_pos = tmp\_pos + 1;  
  }  
  
  for (i=0; i <= num\_elements; i++)  
  {  
    numbers[right] = temp[right];  
    right = right - 1;  
  }  
}