

INSTITUTO TECNOLÓGICO DE ENSENADA

INTELIGENCIA ARTIFICIAL



ALUMNOS:

- Francisco Javier Rochin Acosta
- Edgar Catarino López Ramos

ESPECIALIDAD: Ingeniería En Sistemas Computacionales

GRUPO: 9SA

TRABAJO: Reporte de Practicas

PROFESOR:

- M.I.A. CHAVEZ SANCHEZ GUILLERMO ALEJANDRO

23 DE SEPTIEMBRE DE 2016

METODO BURBUJA

ALGORITMO BURBUJA:

Consiste en tomar desde el primer número de la lista e irlo comparando con todos los demás números de la lista, hasta encontrar su posición, y así con todos los números hasta que la lista esta ordenada.

COMPLEJIDAD EN ESPACIO:

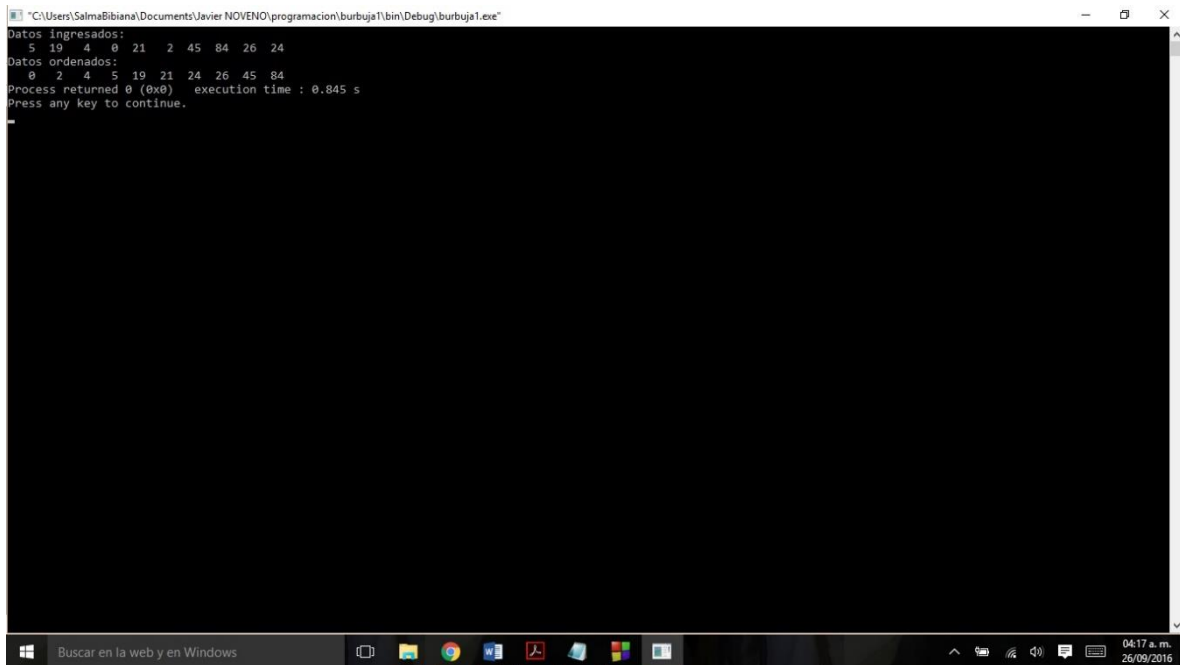
C++: Memoria estática pues las variables se declararon en el algoritmo

```
int a[50],n,i,j,temp = 10 bytes.
```

Python: Memoria dinámica pues depende de cada ejecución del algoritmo.

CAPTURAS DE PANTALLA:

C++

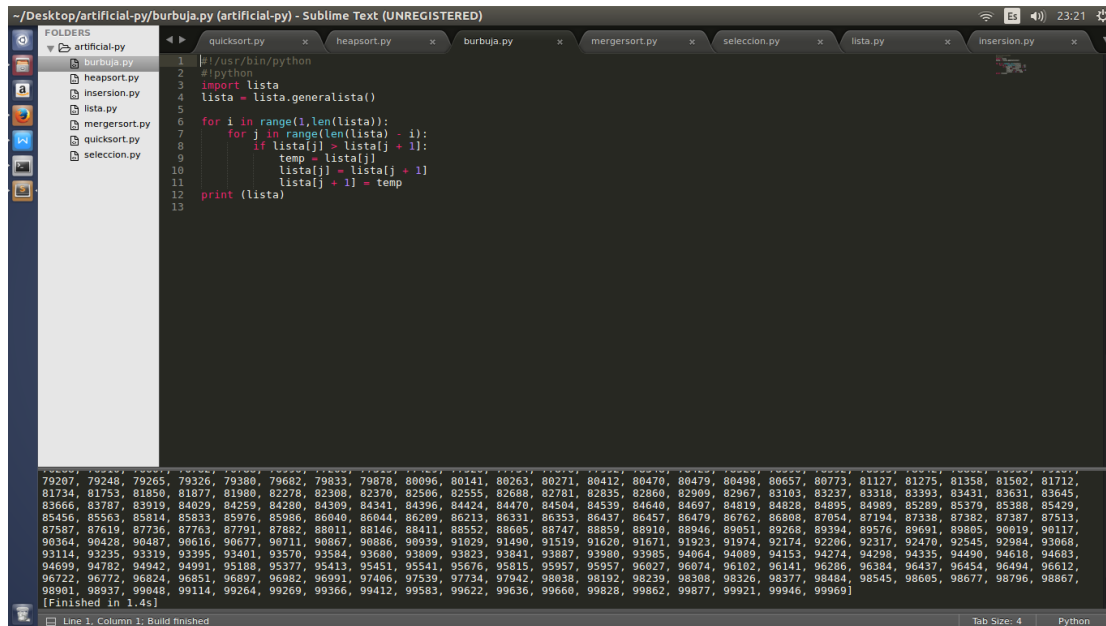


The screenshot shows a Windows command prompt window titled "C:\Users\SalmaBibiana\Documents\Javier NOVENO\programacion\burbuja\bin\Debug\burbuja1.exe". The program output is as follows:

```
Datos ingresados:  
5 19 4 0 21 2 45 84 26 24  
Datos ordenados:  
0 2 4 5 19 21 24 26 45 84  
Process returned 0 (0x0)   execution time : 0.845 s  
Press any key to continue.
```

The Windows taskbar at the bottom shows the search bar with the text "Buscar en la web y en Windows", several application icons, and the system clock displaying "04:17 a. m." and "26/09/2016".

PYTHON



```
~/Desktop/artificial-py/burbuja.py (artificial-py) - Sublime Text (UNREGISTERED)
FOLDERS
  artificial-py
    burbuja.py
    heapsort.py
    insersion.py
    lista.py
    mergesort.py
    quicksort.py
    seleccion.py

1 #!/usr/bin/python
2
3 import lista
4 lista = lista.generalista()
5
6 for i in range(1, len(lista)):
7     for j in range(len(lista) - i):
8         if lista[j] > lista[j + 1]:
9             temp = lista[j]
10            lista[j] = lista[j + 1]
11            lista[j + 1] = temp
12
13 print (lista)

79207, 79248, 79265, 79326, 79380, 79682, 79833, 79878, 80096, 80141, 80263, 80271, 80412, 80470, 80479, 80498, 80657, 80773, 81127, 81275, 81358, 81502, 81712,
81734, 81753, 81850, 81877, 81980, 82278, 82308, 82370, 82506, 82555, 82688, 82781, 82835, 82860, 82909, 82967, 83103, 83237, 83318, 83393, 83431, 83631, 83645,
83666, 83787, 83919, 84029, 84259, 84280, 84309, 84341, 84396, 84424, 84470, 84504, 84539, 84640, 84697, 84819, 84828, 84895, 84989, 85289, 85379, 85388, 85429,
85456, 85563, 85814, 85833, 85976, 85986, 86040, 86044, 86209, 86213, 86331, 86353, 86437, 86457, 86479, 86762, 86808, 87054, 87194, 87338, 87382, 87387, 87513,
87587, 87619, 87736, 87763, 87791, 87882, 88011, 88146, 88411, 88552, 88605, 88747, 88859, 88910, 88946, 89051, 89268, 89394, 89576, 89691, 89805, 90019, 90117,
90364, 90428, 90487, 90610, 90677, 90711, 90867, 90886, 90939, 91029, 91490, 91519, 91620, 91671, 91923, 91974, 92174, 92206, 92317, 92470, 92545, 92584, 93068,
93114, 93235, 93319, 93395, 93401, 93570, 93584, 93680, 93809, 93823, 93841, 93887, 93980, 93985, 94064, 94089, 94153, 94274, 94298, 94335, 94490, 94610, 94683,
94699, 94782, 94942, 94991, 95188, 95377, 95413, 95451, 95541, 95676, 95815, 95957, 95957, 96027, 96074, 96102, 96141, 96286, 96384, 96437, 96454, 96494, 96612,
96722, 96772, 96824, 96851, 96897, 96982, 96991, 97406, 97539, 97734, 97942, 98038, 98192, 98239, 98308, 98326, 98377, 98484, 98545, 98605, 98677, 98796, 98867,
98901, 98937, 99048, 99114, 99264, 99269, 99366, 99412, 99583, 99622, 99636, 99660, 99828, 99862, 99877, 99921, 99946, 99969]
[Finished in 1.4s]

Line 1, Column 1: Build finished
Tab Size: 4 Python
```

ALGORITMO:

```
void bubbleSort (double v[])
```

```
{
```

```
    double tmp;
```

```
    int i,j;
```

```
    int N = v.length;
```

```
    for (i = 0; i < N - 1; i++)
```

```
    {
```

```
        for (j = N - 1; j > i; j--)
```

```
        {
```

```
            if (v[j] < v[j-1])
```

```
            {
```

```
                tmp = v[j];
```

```
                v[j] = v[j-1];
```

```
                v[j-1] = tmp;
```

```
            }
```

```
        }
```

```
    }
```

```
}
```

METODO DE INSERCIÓN

METODO DE INSERCIÓN:

Este método se basa en ir ordenando los números por comparación de sus antecesores o del método que ya existe, es decir se comparan los números nuevos con los anteriores y si el método es mayor se pone antes sino se pone después.

COMPLEJIDAD EN ESPACIO:

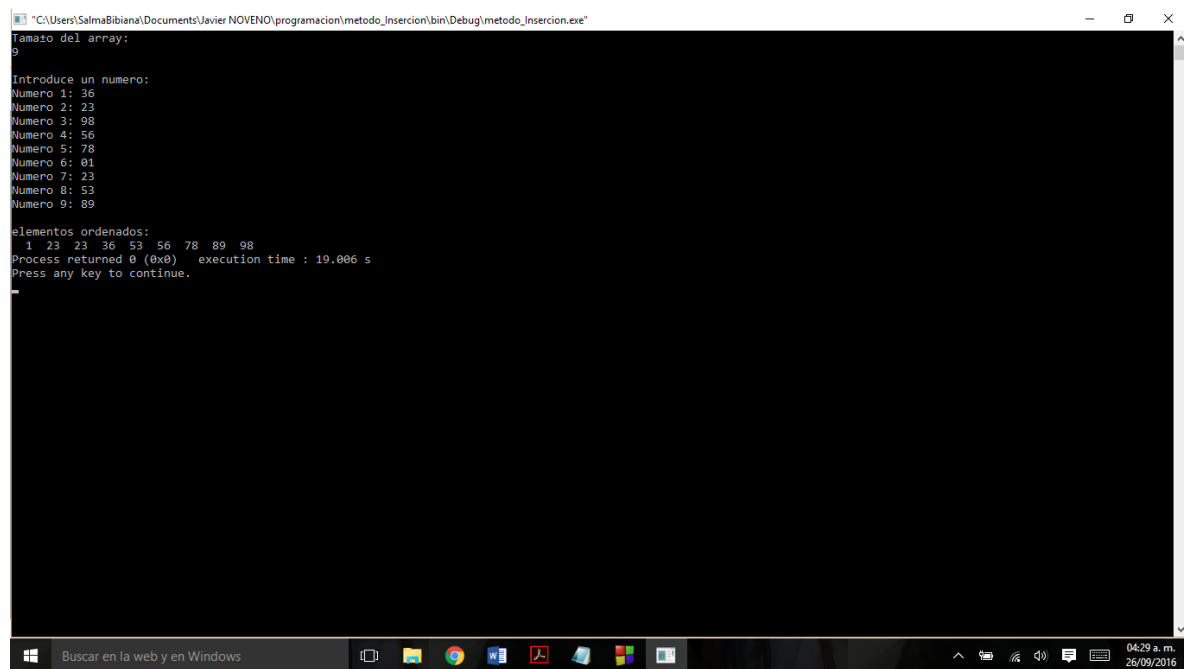
C++: Memoria estatica debido a que las variables se declararon dentro del algoritmo

int A[10],n,i,j,aux = 10 BYTES

Python: Memoria dinámica pues depende de cada ejecución del algoritmo.

CAPTURAS DE PANTALLA:

C++



```
"C:\Users\SalmaBibiana\Documents\Javier NOVENO\programacion\metodo Insercion\bin\Debug\metodo Insercion.exe"
Tamazo del array:
9
Introduce un numero:
Numero 1: 36
Numero 2: 23
Numero 3: 98
Numero 4: 56
Numero 5: 78
Numero 6: 01
Numero 7: 23
Numero 8: 53
Numero 9: 89
elementos ordenados:
1 23 23 36 53 56 78 89 98
Process returned 0 (0x0)   execution time : 19.006 s
Press any key to continue.
```

PYTHON:

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
import lista

lista = lista.generalista()
l=len(lista)
for i in range(0, l-1):
    min=i
    temp=lista[i]
    for j in range(i, l):
        if lista[j] < temp:
            min=j
            temp = lista[j]
            lista[min] = lista[i]
            lista[i] = temp
print lista
```

78936, 79017, 79021, 79088, 79183, 79255, 79602, 79673, 79688, 79705, 79724, 79759, 79892, 79948, 79958, 79985, 79997, 80206, 80494, 80543, 80544, 80626, 80683, 80690, 80786, 81050, 81155, 81267, 81386, 81463, 81510, 81603, 81653, 81723, 81782, 81832, 82327, 82418, 82551, 82606, 82792, 82796, 83008, 83039, 83051, 83066, 83394, 83454, 83656, 83692, 84094, 84170, 84201, 84342, 84400, 84414, 84465, 84537, 84555, 84581, 84592, 84620, 84715, 84725, 84772, 84821, 84903, 85096, 85222, 85284, 85486, 85586, 85604, 85745, 85761, 85810, 85846, 86098, 86126, 86170, 86250, 86264, 86375, 86411, 86462, 86550, 86599, 86600, 86654, 86987, 87067, 87085, 87107, 87178, 87179, 87247, 87400, 87443, 87514, 87568, 87569, 87636, 87835, 87931, 87933, 88141, 88200, 88434, 88526, 88548, 88672, 88903, 89051, 89155, 89222, 89271, 89308, 89362, 89401, 89536, 89688, 89855, 90147, 90219, 90264, 90328, 90458, 90562, 90592, 90646, 90887, 90931, 90976, 90986, 90988, 91005, 91038, 91050, 91086, 91119, 91129, 91178, 91302, 91453, 91499, 91688, 91716, 91727, 91848, 91892, 91906, 92069, 92188, 92189, 92202, 92257, 92271, 92404, 92584, 92617, 92638, 92738, 93168, 93202, 93314, 93322, 93568, 93586, 93862, 94006, 94080, 94106, 94140, 94287, 94324, 94444, 94489, 94517, 94564, 94830, 95144, 95150, 95161, 95299, 95405, 95488, 95499, 95509, 95567, 95590, 95599, 95676, 95719, 95751, 95816, 95862, 95903, 95916, 96027, 96031, 96107, 96220, 96376, 96817, 97108, 97111, 97255, 97327, 97591, 97679, 97826, 97836, 98273, 98464, 98936, 99123, 99412, 99414, 99524, 99730, 99881, 99906, 99917]

Line 15, Column 12: Build finished

Tab Size: 4 Python

ALGORITMO:

```
void insertionSort (double v[])
{
    double tmp;
    int i, j;
    int N = v.length;
    for (i=1; i<N; i++)
    {
        tmp = v[i];
        for (j=i; (j>0) && (tmp<v[j-1]); j--)
        {
            v[j] = v[j-1];
        }
        v[j] = tmp;
    }
}
```

METODO SELECCION

ALGORITMO POR SELECCION:

Este algoritmo se basa en tomar el número menor del arreglo y posicionarlo al principio e ir comparando así con el segundo, el tercer, cuarto y n números posicionándolos después del número menor del arreglo.

COMPLEJIDAD EN ESPACIO:

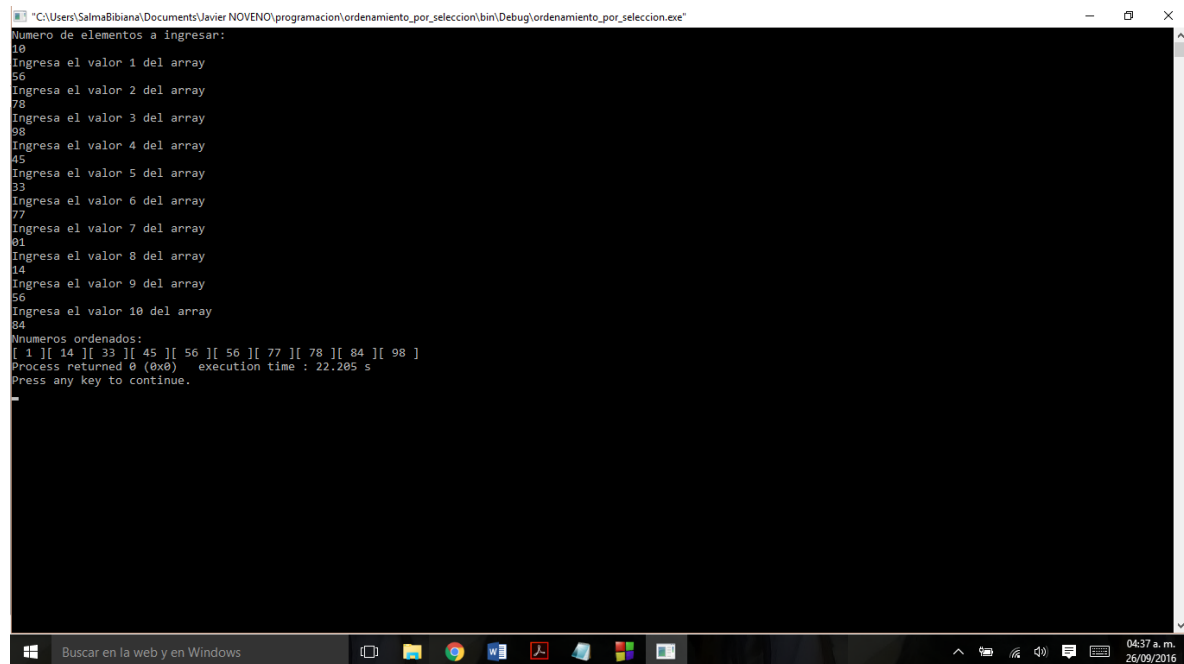
C++: Memoria estática pues las variables se declararon en el algoritmo

```
int k, menor, i, j= 8 bytes.
```

Python: Memoria dinámica pues depende de cada ejecución del algoritmo.

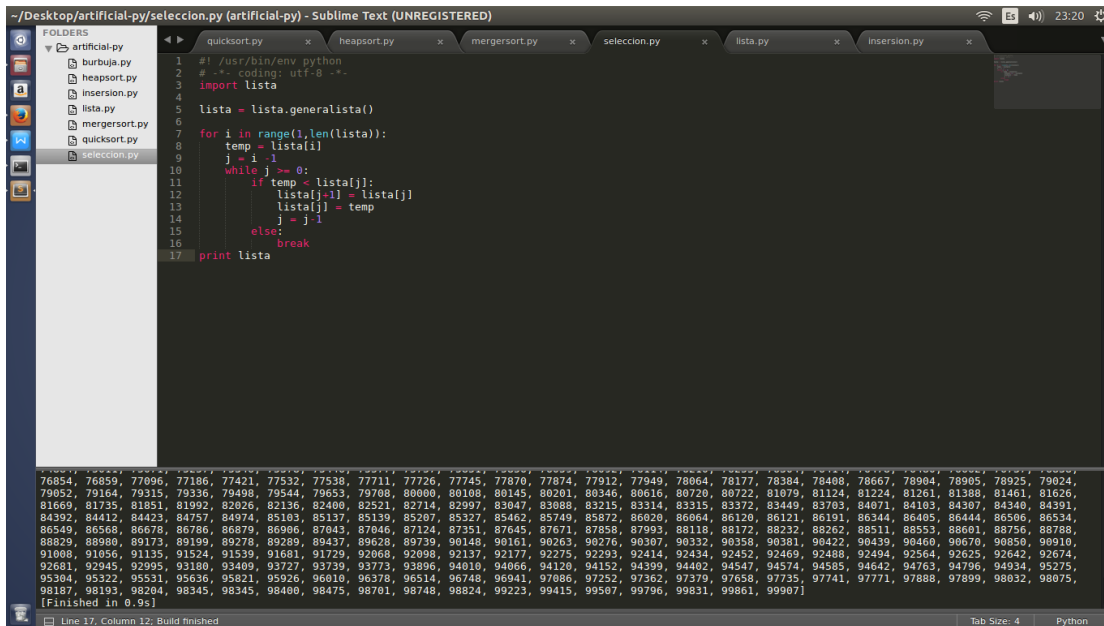
CAPTURAS DE PANTALLA:

C++



```
"C:\Users\SalmaBibiana\Documents\Javier NOVENO\programacion\ordenamiento_por_seleccion\bin\Debug\ordenamiento_por_seleccion.exe"
Numero de elementos a ingresar:
10
Ingresa el valor 1 del array
56
Ingresa el valor 2 del array
78
Ingresa el valor 3 del array
98
Ingresa el valor 4 del array
45
Ingresa el valor 5 del array
33
Ingresa el valor 6 del array
77
Ingresa el valor 7 del array
81
Ingresa el valor 8 del array
14
Ingresa el valor 9 del array
56
Ingresa el valor 10 del array
84
Numeros ordenados:
[ 1 ][ 14 ][ 33 ][ 45 ][ 56 ][ 56 ][ 77 ][ 78 ][ 84 ][ 98 ]
Process returned 0 (0x0)   execution time : 22.285 s
Press any key to continue.
```

PYTHON



```
~/Desktop/artificial-py/seleccion.py (artificial-py) - Sublime Text (UNREGISTERED)
FOLDERS
  artificial-py
    burbuja.py
    heapsort.py
    insertion.py
    lista.py
    mergesort.py
    quicksort.py
    seleccion.py

1  #!/usr/bin/env python
2  #-*- coding: utf-8 -*-
3  import lista
4
5  lista = lista.generalista()
6
7  for i in range(1, len(lista)):
8      temp = lista[i]
9      j = i - 1
10     while j >= 0:
11         if temp < lista[j]:
12             lista[j+1] = lista[j]
13             lista[j] = temp
14             j = j - 1
15         else:
16             break
17     print lista

76854, 76859, 77096, 77186, 77421, 77532, 77538, 77711, 77726, 77745, 77870, 77874, 77912, 77949, 78064, 78177, 78384, 78408, 78667, 78904, 78905, 78925, 79024,
79052, 79164, 79315, 79336, 79498, 79544, 79653, 79708, 80008, 80108, 80145, 80201, 80346, 80616, 80720, 80722, 81079, 81124, 81224, 81261, 81388, 81461, 81626,
81660, 81735, 81851, 81922, 82026, 82136, 82400, 82521, 82714, 82997, 83047, 83086, 83215, 83314, 83315, 83372, 83449, 83703, 84071, 84103, 84307, 84340, 84391,
84392, 84412, 84423, 84757, 84974, 85103, 85137, 85139, 85207, 85327, 85462, 85749, 85872, 86020, 86064, 86120, 86121, 86191, 86344, 86405, 86444, 86506, 86534,
86549, 86568, 86678, 86786, 86879, 86906, 87043, 87046, 87124, 87351, 87645, 87671, 87858, 87993, 88118, 88172, 88232, 88262, 88511, 88553, 88601, 88756, 88788,
88829, 88908, 89173, 89199, 89278, 89289, 89437, 89628, 89739, 90148, 90161, 90263, 90276, 90307, 90332, 90358, 90381, 90422, 90439, 90460, 90670, 90850, 90910,
91000, 91056, 91135, 91524, 91539, 91681, 91729, 92068, 92098, 92137, 92177, 92275, 92293, 92414, 92434, 92452, 92469, 92488, 92494, 92564, 92625, 92642, 92674,
92681, 92945, 92995, 93180, 93409, 93727, 93739, 93773, 93896, 94010, 94066, 94120, 94152, 94399, 94402, 94547, 94574, 94585, 94642, 94763, 94796, 94934, 95275,
95304, 95322, 95531, 95636, 95821, 95926, 96010, 96378, 96514, 96748, 96941, 97086, 97252, 97362, 97379, 97658, 97735, 97741, 97771, 97888, 97899, 98032, 98075,
98187, 98193, 98204, 98345, 98345, 98400, 98475, 98701, 98748, 98824, 99223, 99415, 99507, 99796, 99831, 99861, 99907]
[Finished in 0.9s]
```

ALGORITMO:

```
void insertionSort (double v[])
{
    double tmp;
    int i, j;
    int N = v.length;
    for (i=1; i<N; i++)
    {
        tmp = v[i];
    }
    for (j=i; (j>0) && (tmp<v[j-1]); j--)
    {
        v[j] = v[j-1];
        v[j] = tmp;
    }
}
```

METODO QUICKSORT

ALGORITMO QUICKSORT:

Este algoritmo consiste en tomar un numero como pivote y acomodar los números dependiendo de si son mayores al pivote o menores al pivote, en caso de ser mayores irán al lado derecho y si son menores irán al lado izquierdo, y así se repite el algoritmo para realizar un árbol binario el cual tendrá de un lado los menores y del otro lado los mayores al pivote.

COMPLEJIDAD EN ESPACIO:

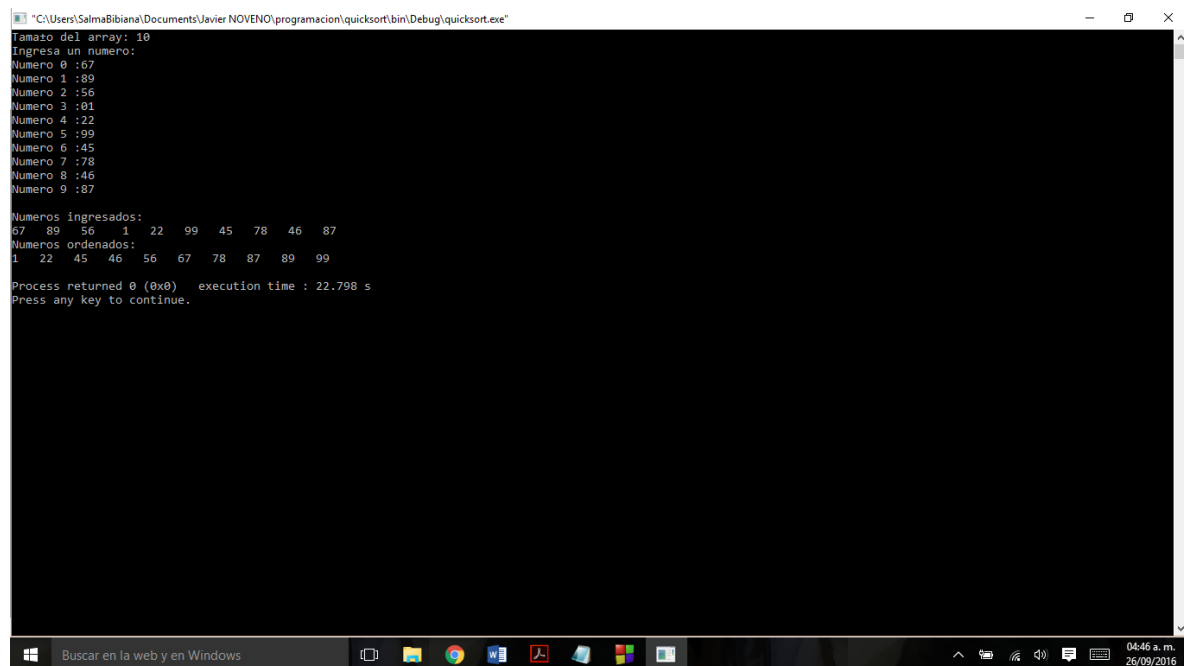
C++: Memoria estática pues las variables se declararon en el algoritmo

```
int cen, pri, ult, i, j, piv;= 12 bytes.
```

Python: Memoria dinámica pues depende de cada ejecución del algoritmo.

CAPTURAS DE PANTALLA:

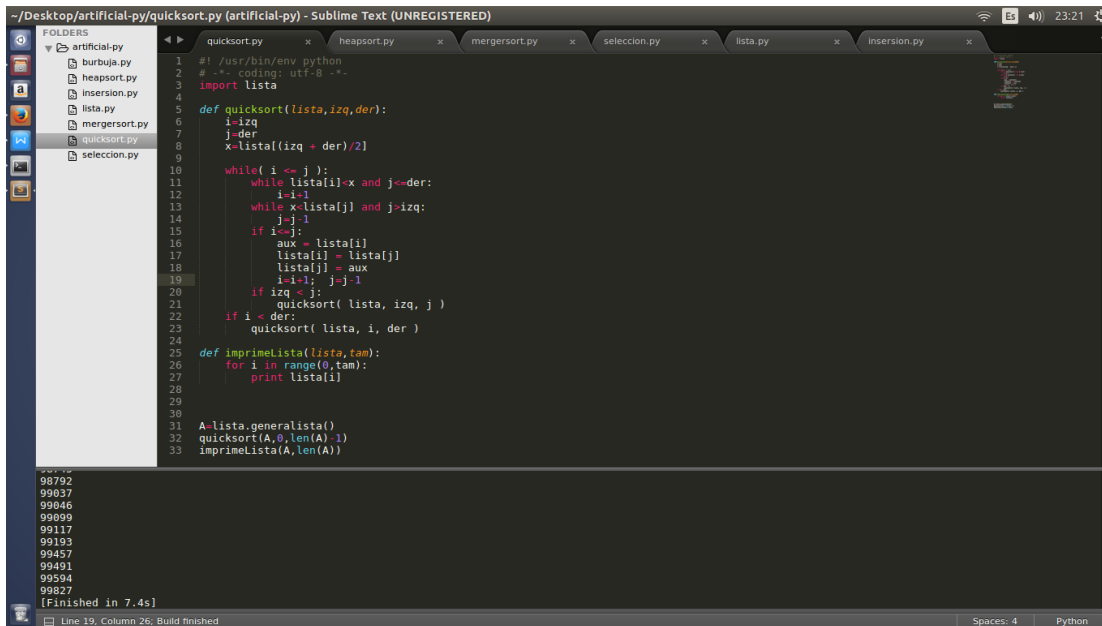
C++



```
"C:\Users\SalmaBibiana\Documents\Javier NOVENO\programacion\quicksort\bin\Debug\quicksort.exe"
Tamano del array: 10
Ingresa un numero:
Numero 0 :67
Numero 1 :89
Numero 2 :56
Numero 3 :01
Numero 4 :22
Numero 5 :99
Numero 6 :45
Numero 7 :78
Numero 8 :46
Numero 9 :87

Numeros ingresados:
67 89 56 1 22 99 45 78 46 87
Numeros ordenados:
1 22 45 46 56 67 78 87 89 99
Process returned 0 (0x0)   execution time : 22.798 s
Press any key to continue.
```


PYTHON



```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
import lista

def quicksort(lista, izq, der):
    i=izq
    j=der
    x=lista[(izq + der)/2]

    while i <= j:
        while lista[i]>x and j<=der:
            i=i+1
        while x<lista[j] and j>=izq:
            j=j-1
        if i<=j:
            aux = lista[i]
            lista[i] = lista[j]
            lista[j] = aux
            i=i+1; j=j-1
        if izq < j:
            quicksort( lista, izq, j )
        if i < der:
            quicksort( lista, i, der )

def imprimeLista(lista,tam):
    for i in range(0,tam):
        print lista[i]

A=lista.generalista()
quicksort(A,0,len(A)-1)
imprimeLista(A,len(A))
```

ALGORITMO:

```
void quicksort (double v[], int izda, int dcha)
{
    int pivote; // Posición del pivote
    if (izda<dcha)
    {
        pivote = partir (v, izda, dcha);
        quicksort (v, izda, pivote-1);
        quicksort (v, pivote+1, dcha);
    }
}
```

METODO MERGESORT

ALGORITMO MERGESORT:

Consiste en ir dividiendo la lista hasta la mínima cantidad e irlos comparando de manera inversa para al final ir juntando las sublistas e ir formando la lista ordenada.

COMPLEJIDAD EN ESPACIO:

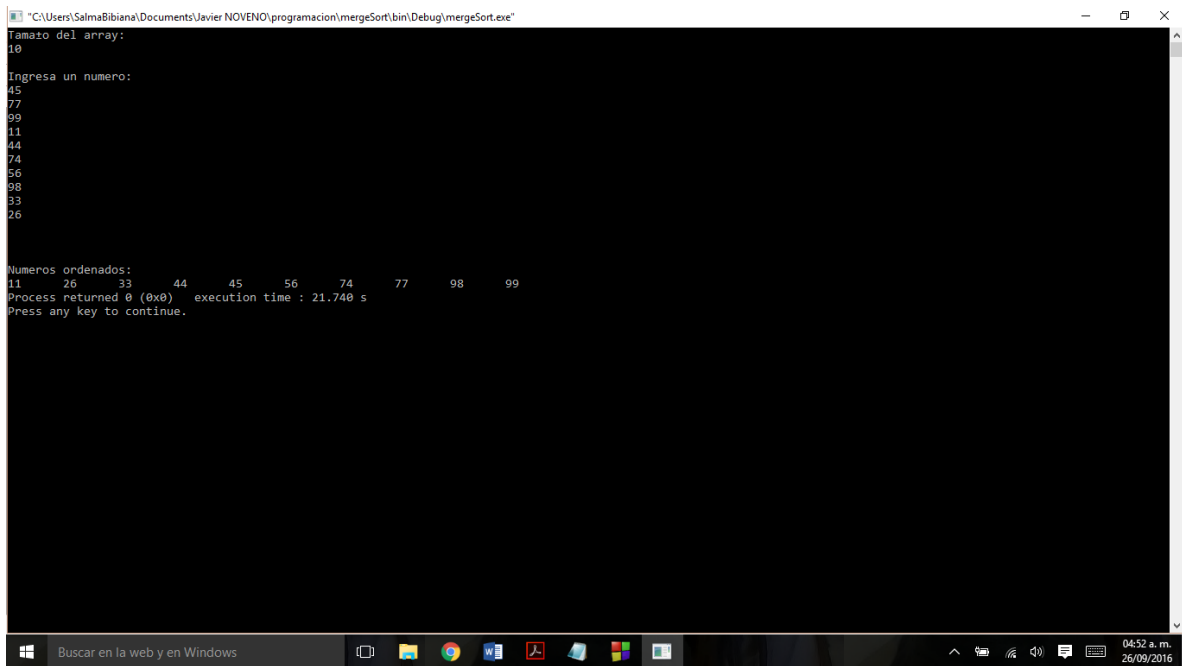
C++: Memoria estática pues las variables se declararon en el algoritmo

```
Int izq, med, der, h,i,j,b[50],k = 16 bytes
```

Python: Memoria dinámica pues depende de cada ejecución del algoritmo.

CAPTURAS DE PANTALLA:

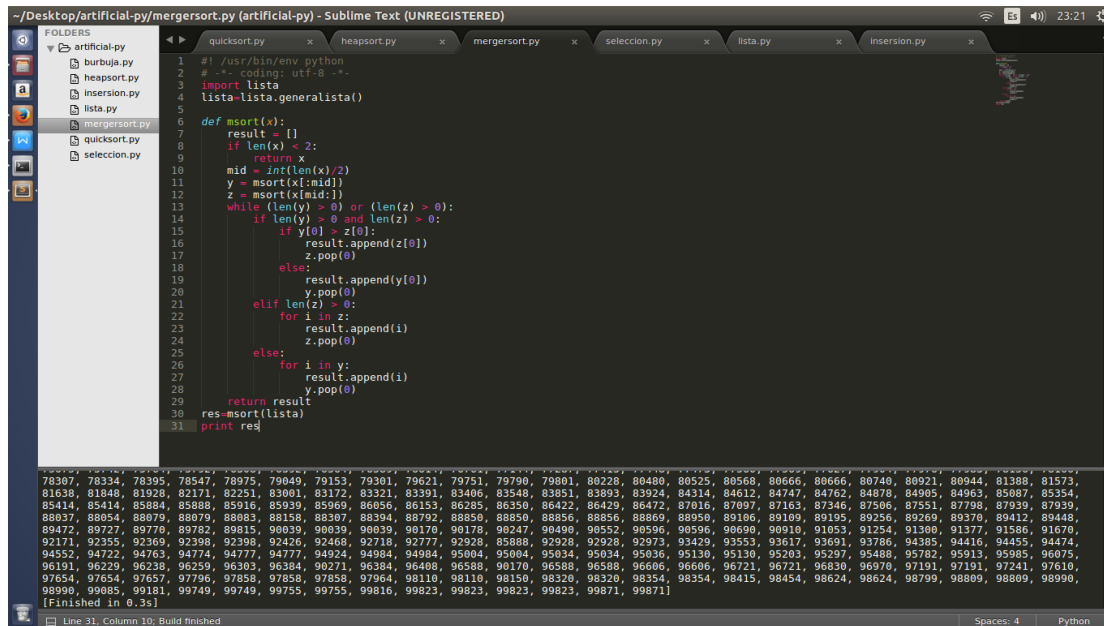
C++



```
"C:\Users\SalmaBibiana\Documents\Javier NOVENO\programacion\mergeSort\bin\Debug\mergeSort.exe"
Tamazo del array:
18
Ingresa un numero:
45
77
99
11
44
74
56
98
33
26

Numeros ordenados:
11 26 33 44 45 56 74 77 98 99
Process returned 0 (0x0)   execution time : 21.740 s
Press any key to continue.
```

PYTHON



```
~/Desktop/artificial-py/mergersort.py (artificial-py) - Sublime Text (UNREGISTERED)
FOLDERS
  artificial-py
    burbuja.py
    heapsort.py
    insersion.py
    lista.py
    mergersort.py
    quicksort.py
    seleccion.py

1  #!/usr/bin/env python
2  # -*- coding: utf-8 -*-
3  import lista
4  lista=lista.generalista()
5
6
7  def msort(x):
8      result = []
9      if len(x) < 2:
10         return x
11     mid = int(len(x)/2)
12     y = msort(x[:mid])
13     z = msort(x[mid:])
14     while (len(y) > 0) or (len(z) > 0):
15         if len(y) > 0 and len(z) > 0:
16             if y[0] > z[0]:
17                 result.append(z[0])
18                 z.pop(0)
19             else:
20                 result.append(y[0])
21                 y.pop(0)
22         elif len(z) > 0:
23             for i in z:
24                 result.append(i)
25                 z.pop(0)
26         else:
27             for i in y:
28                 result.append(i)
29                 y.pop(0)
30     return result
31 res=msort(lista)
32 print res

78307, 78334, 78395, 78547, 78975, 79049, 79153, 79301, 79621, 79751, 79790, 79801, 80228, 80480, 80525, 80568, 80666, 80666, 80740, 80921, 80944, 81388, 81573,
81638, 81848, 81928, 82171, 82251, 83001, 83172, 83321, 83391, 83406, 83548, 83851, 83893, 83924, 84314, 84612, 84747, 84762, 84878, 84905, 84963, 85087, 85354,
85414, 85414, 85884, 85888, 85916, 85939, 85969, 86056, 86153, 86285, 86350, 86422, 86429, 86472, 87016, 87097, 87163, 87346, 87506, 87551, 87796, 87939, 87939,
88037, 88054, 88079, 88079, 88083, 88158, 88307, 88394, 88792, 88850, 88850, 88856, 88869, 88950, 89186, 89109, 89195, 89256, 89269, 89370, 89412, 89448,
89472, 89727, 89770, 89782, 89815, 90039, 90039, 90039, 90170, 90178, 90247, 90490, 90552, 90596, 90596, 90690, 90910, 91053, 91254, 91300, 91377, 91586, 91670,
92171, 92355, 92369, 92398, 92398, 92426, 92468, 92718, 92777, 92928, 92928, 92928, 92973, 93429, 93553, 93617, 93691, 93786, 94385, 94416, 94455, 94474,
94552, 94722, 94763, 94774, 94777, 94777, 94924, 94984, 94984, 95004, 95004, 95034, 95036, 95130, 95130, 95203, 95297, 95480, 95782, 95913, 95985, 96075,
96191, 96229, 96238, 96259, 96303, 96384, 96384, 96384, 96408, 96588, 96588, 96588, 96588, 96606, 96606, 96721, 96721, 96830, 96970, 97191, 97191, 97241, 97610,
97654, 97654, 97657, 97796, 97858, 97858, 97858, 97964, 98110, 98110, 98150, 98320, 98320, 98354, 98354, 98415, 98454, 98624, 98624, 98799, 98809, 98809, 98990,
98990, 98985, 99181, 99749, 99749, 99755, 99755, 99816, 99823, 99823, 99823, 99823, 99871, 99871]
[Finished in 0.3s]
```

ALGORITMO:

```
void mergeSort(int izq,int der)
```

```
{
```

```
    int med;
```

```
    if(izq<der)
```

```
    {
```

```
        med=(izq+der)/2;
```

```
        mergeSort(izq,med);
```

```
        mergeSort(med+1,der);
```

```
        merge(izq,med,der);
```

```
    }
```

```
}
```

METODO HEAPSORT

ALGORITMO HEAPSORT:

consiste en meter la lista en un árbol binario e irlo recorriendo de manera que ni un nodo hijo sea mayor al nodo padre, al terminar eso se va sacando la raíz y se sustituye por el ultimo nodo o hoja, y se vuelve a comprobar la regla de que ningún hijo sea mayor al padre.

COMPLEJIDAD EN ESPACIO:

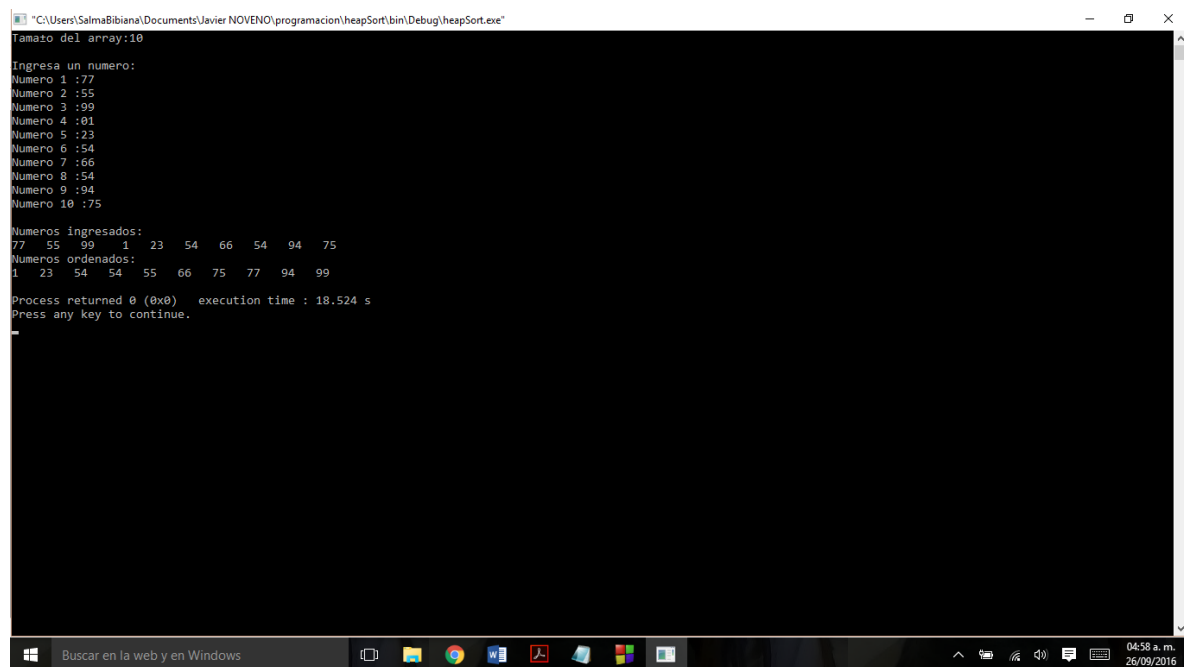
C++: Memoria estática pues las variables se declararon en el algoritmo

```
int A[T],j,piv,tem,i,k,n;= 14 bytes
```

Python: Memoria dinámica pues depende de cada ejecución del algoritmo.

CAPTURAS DE PANTALLA:

C++



```
"C:\Users\SalmaBibiana\Documents\Javier NOVENO\programacion\heapSort\bin\Debug\heapSort.exe"
Tamazo del array:10
Ingresa un numero:
Numero 1 :77
Numero 2 :55
Numero 3 :99
Numero 4 :01
Numero 5 :23
Numero 6 :54
Numero 7 :66
Numero 8 :54
Numero 9 :94
Numero 10 :75
Numeros ingresados:
77 55 99 1 23 54 66 54 94 75
Numeros ordenados:
1 23 54 54 55 66 75 77 94 99
Process returned 0 (0x0) execution time : 18.524 s
Press any key to continue.
```

```
~/Desktop/artificial-py/heapsort.py (artificial-py) - Sublime Text (UNREGISTERED)
FOLDERS
  artificial-py
    burbuja.py
    heapsort.py
    insercion.py
    lista.py
    mergesort.py
    quicksort.py
    seleccion.py

1  #!/usr/bin/env python
2  # -*- coding: utf-8 -*-
3  import lista
4  lista=lista.generallista()
5
6
7  def HeapSort(A):
8      def heapify(A):
9          start = (len(A) - 2) / 2
10         while start >= 0:
11             siftDown(A, start, len(A) - 1)
12             start -= 1
13         def siftDown(A, start, end):
14             root = start
15             while root * 2 + 1 <= end:
16                 child = root * 2 + 1
17                 if child + 1 <= end and A[child] < A[child + 1]:
18                     child += 1
19                 if child <= end and A[root] < A[child]:
20                     A[root], A[child] = A[child], A[root]
21                     root = child
22                 else:
23                     return
24             heapify(A)
25         end = len(A) - 1
26         while end > 0:
27             A[end], A[0] = A[0], A[end]
28             siftDown(A, 0, end - 1)
29             end -= 1
30     HeapSort(lista)
31     print lista

78888, 79010, 79358, 79581, 79680, 79708, 79930, 80000, 80221, 80290, 80467, 80491, 80492, 80495, 80617, 80666, 80711, 80726, 80948, 80948, 81014, 81036, 81059,
81106, 81264, 81379, 81477, 81549, 81717, 81722, 81789, 82154, 82299, 82338, 82468, 82536, 82564, 82609, 82702, 82729, 82954, 82967, 83137, 83235, 83244, 83356,
83402, 83478, 83535, 83540, 83597, 83703, 84053, 84079, 84096, 84213, 84226, 84234, 84301, 84308, 84330, 84470, 84540, 84800, 85285, 85291, 85437, 85540, 85624,
85782, 85919, 85942, 86037, 86131, 86169, 86210, 86224, 86394, 86516, 86535, 86641, 86641, 86687, 86695, 86699, 86701, 86754, 86915, 87011, 87114, 87137, 87176,
87397, 87424, 87525, 87729, 87741, 87922, 87988, 88031, 88077, 88186, 88187, 88383, 88621, 88668, 88677, 88980, 89107, 89223, 89271, 89350, 89437, 89648, 89776,
89901, 90039, 90106, 90150, 90268, 90346, 90623, 90758, 90780, 90780, 90823, 90917, 91043, 91108, 91114, 91186, 91494, 91568, 91703, 91708, 91726, 91794, 91876,
92192, 92217, 92443, 92598, 92613, 92681, 92708, 92787, 92885, 92990, 93004, 93151, 93208, 93252, 93386, 93393, 93455, 93464, 93500, 93564, 93589, 93752, 93844,
93924, 94149, 94172, 94296, 94357, 94513, 94658, 94883, 94895, 94942, 94998, 95106, 95152, 95177, 95412, 95525, 95686, 95787, 95984, 96101, 96267, 96292, 96423,
96553, 96666, 96791, 96995, 97199, 97218, 97277, 97777, 97807, 97811, 97866, 97975, 98053, 98091, 98098, 98189, 98267, 98289, 98364, 98373, 98408, 98548, 98688,
98769, 98778, 99003, 99092, 99273, 99475, 99529, 99542, 99645, 99654, 99675, 99730, 99919, 99931, 99982]
[Finished in 0.2s]
```

ALGORITMO:

```
for(k=n;k>0;k--)
{
    for(i=1;i<=k;i++)
    {
        piv=A[i];
        j=i/2;
        while(j>0 && A[j]<piv)
        {
            A[i]=A[j];
            i=j;
            j=j/2;
        }
        A[i]=piv;
    }
    tem=A[1];
    A[1]=A[k];
    A[k]=tem;
}
```

DIAGRAMA DE FLUJO: METODO BURBUJA

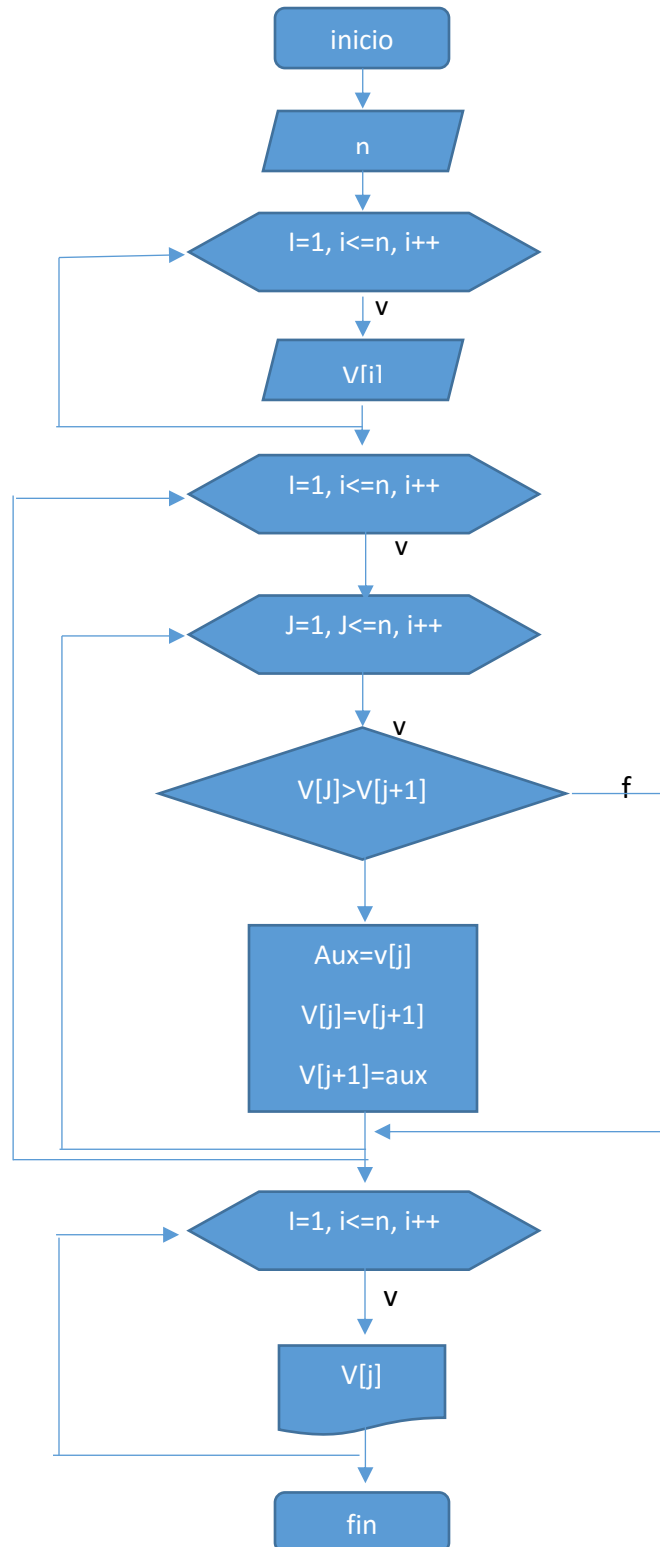


DIAGRAMA DE FLUJO: METODO SELECCIÓN

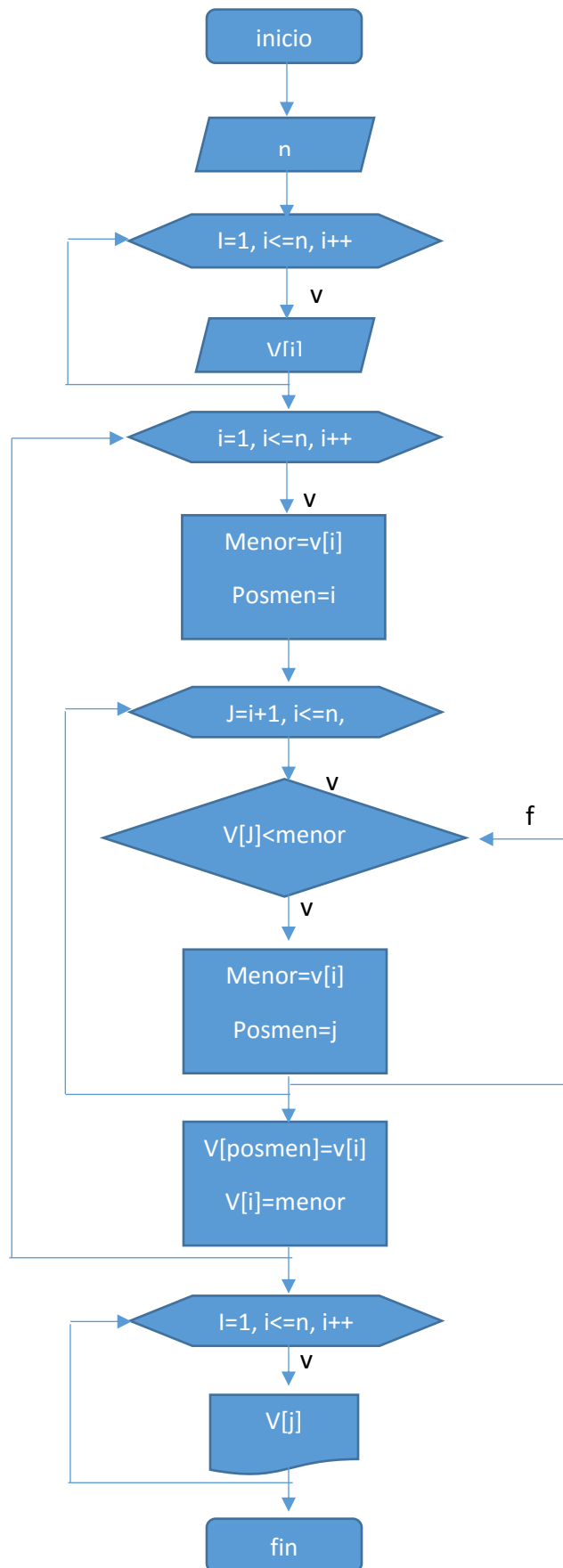


DIAGRAMA DE FLUJO: METODO INSERCIÓN

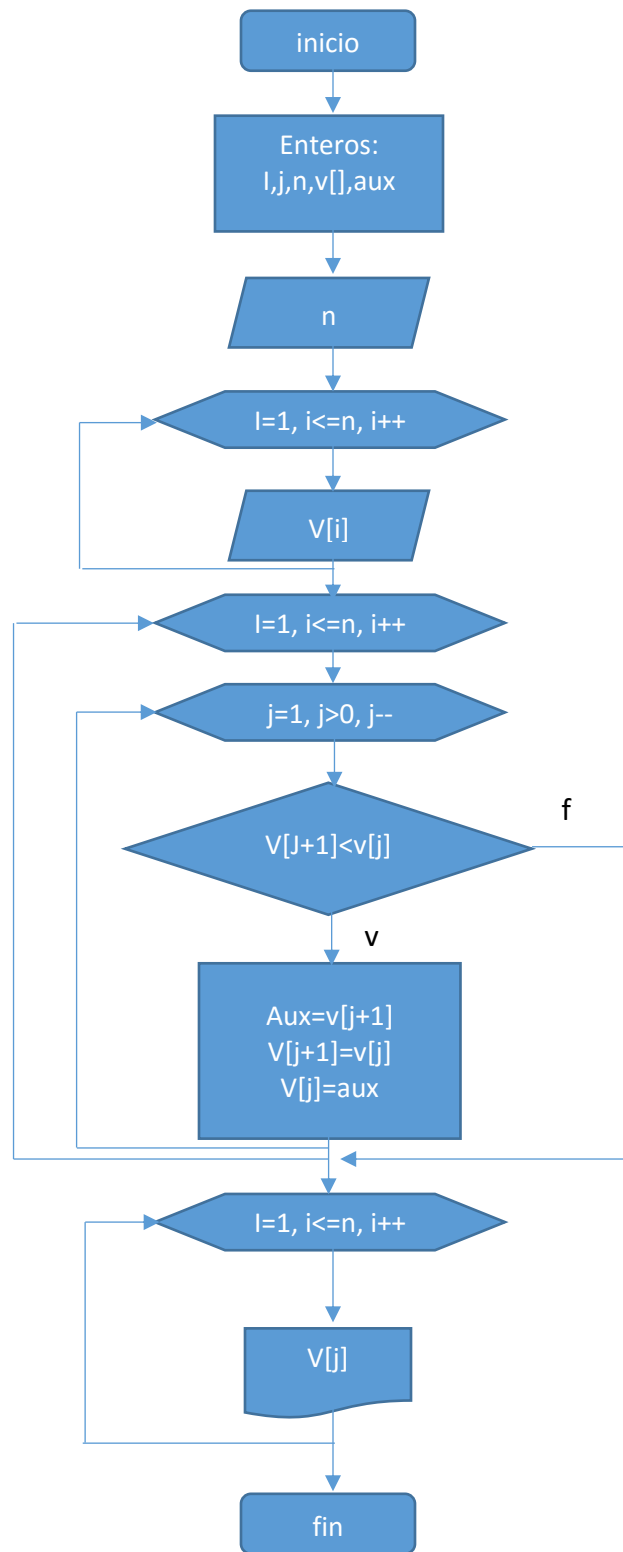


DIAGRAMA DE FLUJO: METODO MERGESORT

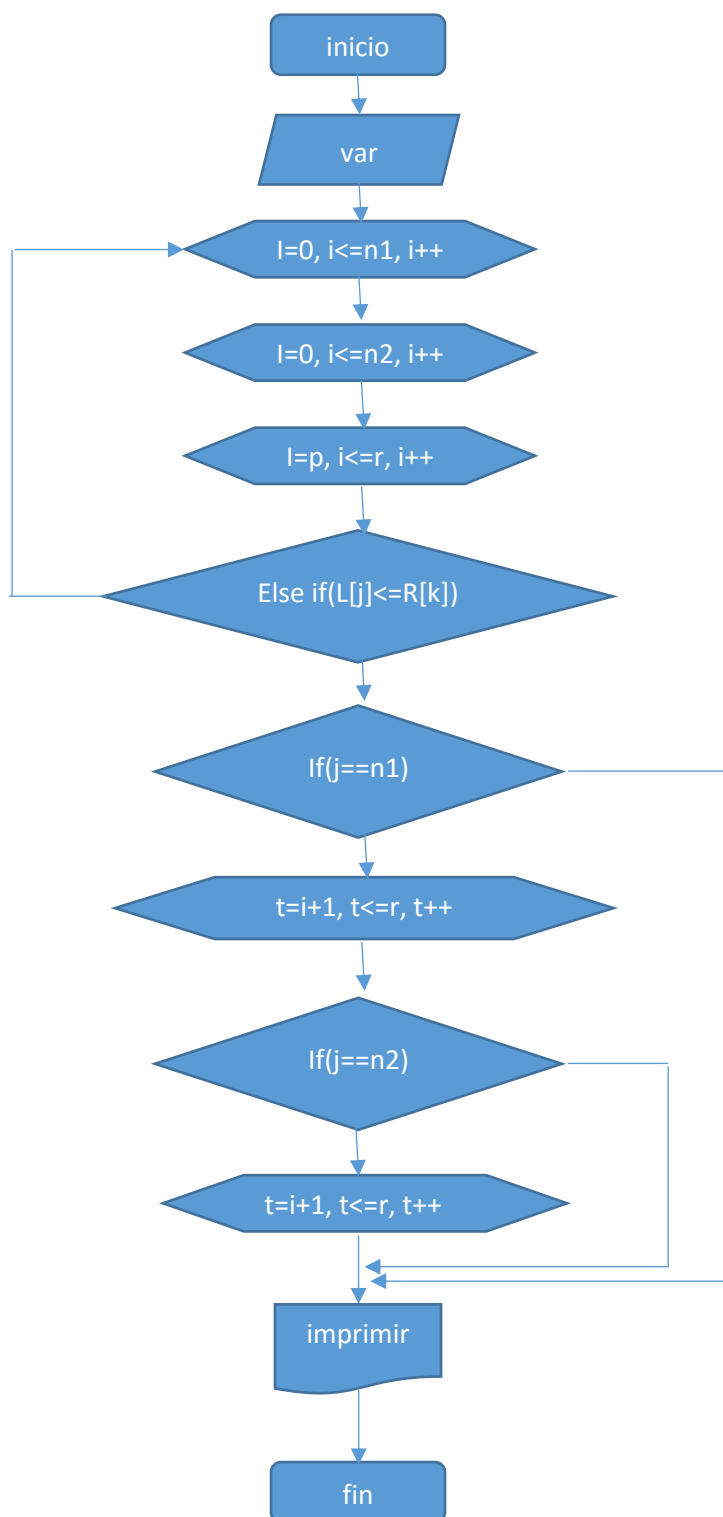
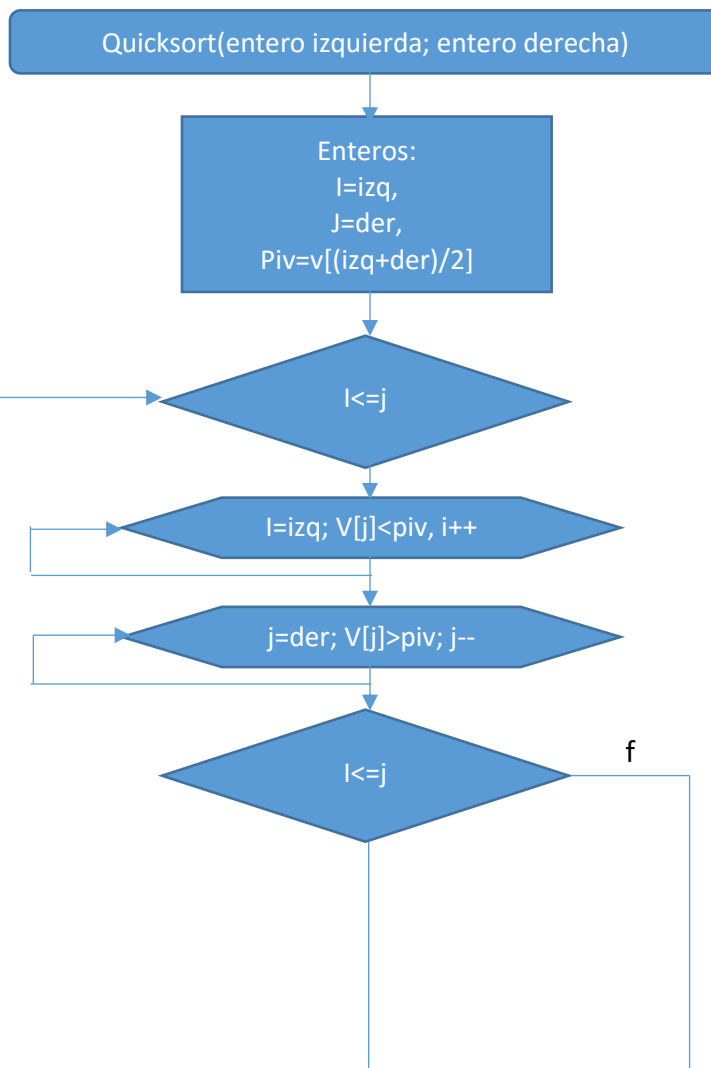
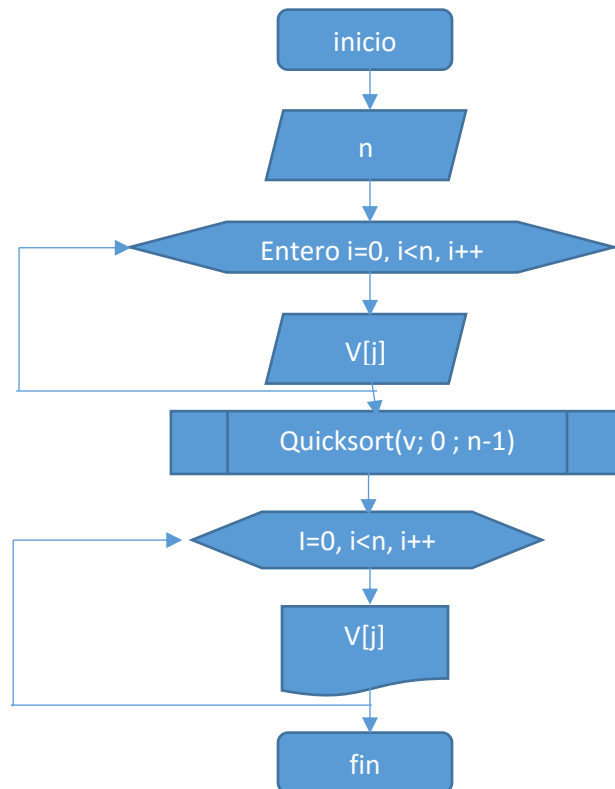
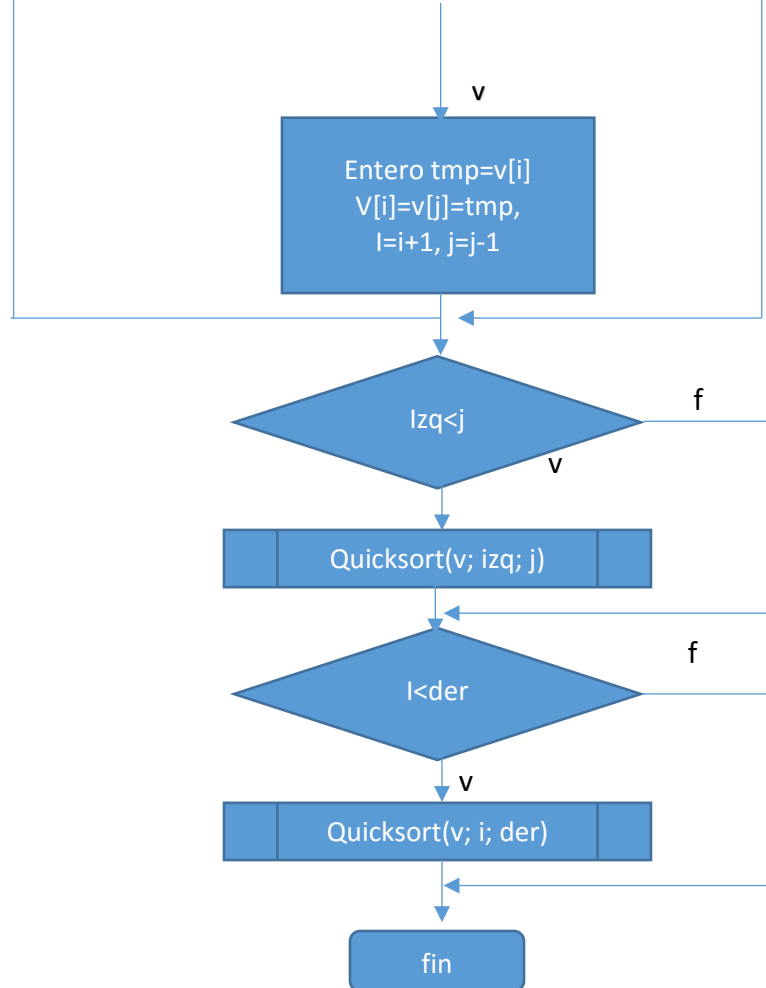


DIAGRAMA DE FLUJO: METODO QUICKSORT





GRAFICA DE TIEMPOS

Algoritmo	Python	C++	N datos python	N datos C++
Selección	0.9 Seg	22.205 Seg	1000	10
Insertion	0.9 Seg	19.006 Seg	1000	10
burbuja	2.5 Seg	0.845 Seg	1000	10
heapsort	0.2 Seg	18.524 Seg	1000	10
mergesort	0.3 Seg	21.740 Seg	1000	10
Quicksort	7.4 Seg	22.798 Seg	1000	10

COMPLEJIDAD:

Algoritmo	Operaciones máximas
Burbuja	$\Omega(n^2)$
Insertión	$\Omega(n^2/4)$
Selección	$\Omega(n^2)$
ShellSort	$\Omega(n \log^2 n)$
MergeSort	$\Omega(n \log n)$
QuickSort	$\Omega(n^2)$ en el peor de los casos y $\Omega(n \log n)$ en el promedio de los casos.

CONCLUSION

Método de la burbuja:

Se basa en comparar los elementos del arreglo e intercambiarlos si su posición actual o inicial es contraria inversa a la que queremos. No es muy eficiente para ordenar listas grandes.

Es fácil de entender y bueno para ordenar una lista que no tenga muchos números.

Selección:

Los métodos de ordenación por selección se basan en dos principios básicos:

Seleccionar el elemento más pequeño (o más grande) del arreglo.

Colocarlo en la posición más baja (o más alta) del arreglo.

Este método no es muy eficiente para ordenar listas grandes, es muy lento.

A lo que vi al estudiar este método, a diferencia del método de la burbuja, en este método el elemento más pequeño (o más grande) es el que se coloca en la posición final que le corresponde.

Inserción:

No es eficiente para ordenar listas grandes, muy lento.

Este método consiste en insertar los elementos no ordenados del arreglo en subarreglos del mismo que ya estén ordenados. Dependiendo del método elegido para encontrar la posición de inserción tendremos distintas versiones del método de inserción.

Quicksort:

Considero el método de la burbuja como el peor de todos. No es muy eficiente y el método quicksort es una mejora de burbuja. Tiene la idea de dividir una lista grande de números subdividiéndola en partes más pequeñas con un orden entre ellas.

Es eficiente y funciona muy bien, no consume muchos recursos y leí que está estudiado muy bien matemáticamente. Se comprobó que es muy eficiente. Una ventaja muy buena que tiene y de las que hablamos en el salón de clase que es importante, es que su bucle interno es extremadamente corto.

Método Shell o shell sort:

Fue uno de los primeros métodos en romper la barrera del tiempo cuadrático.

Hace comparaciones e intercambios entre elementos no adyacentes, los más apartados. Compara elementos que están distantes, y esta distancia disminuye en la medida que el algoritmo avanza hasta que se comparan los elementos adyacentes del array.

MergeSort:

Este algoritmo fusiona dos arreglos y los ordena en un tercero.

En mi investigación sobre este algoritmo pude averiguar que es un algoritmo recursivo (de tipo divide y vencerás) cuyo tiempo de ejecución es de en el peor caso, y el número de comparaciones usadas es cercana al óptimo.