

INSTITUTO POLITÉCNICO NACIONAL ESCUELA
SUPERIOR DE CÓMPUTO



Análisis de Algoritmos

Práctica Sesión 2 Algoritmos Básicos

Edgar Adrián Nava Romo

Maestra: Sandra Díaz Santiago
Grupo: 3CM3

1. Merge:

Código Fuente:

```
1.  /Complexity O(A+B)
2.  void mergeArrays(int arr1[], int arr2[], int n1, int n2, int arr3[]){
3.      int i = 0, j = 0, k = 0;
4.      // Traverse both array
5.      /* Check if current element of first
6.      array is smaller than current element
7.      of second array. If yes, store first
8.      array element and increment first array
9.      index. Otherwise do same with second array*/
10.     while (i < n1 && j < n2){
11.         if (arr1[i] < arr2[j]) arr3[k++] = arr1[i++];
12.         else arr3[k++] = arr2[j++];
13.     }
14.     // Store remaining elements of first array
15.     while (i < n1)
16.         arr3[k++] = arr1[i++];
17.     // Store remaining elements of second array
18.     while (j < n2)
19.         arr3[k++] = arr2[j++];
20. }
```

Ejemplo:

A = [8, 10, 12]

B = [7, 9, 11]

- Se crea un arreglo llamado arr3 con la suma de arr1 + arr2
- Simultáneamente se comparan y checamos el elemento más pequeño en los dos arreglos en la posición actual e insertamos el más pequeño en arr3.
- Depende el caso se rellenan los números sobrantes de los arreglos al final de arr3

Paso 1: Insertamos 7 en arr3 en su primer posición y avanzamos en arr2

arr1 [8, 10, 12]	arr2 [7, 9, 11]
^	^
arr3 [*, *, *, *, *, *]	
^	

Paso 2: Insertamos el número 8 en arr3 en su segunda posición y avanzamos en arr1

arr1 [8, 10, 12]	arr2 [7, 9, 11]
^	^
arr3 [7, *, *, *, *, *]	
^	

Paso 3: Insertamos el número 9 en arr3 en su tercera posición y avanzamos en arr2

```
arr1 [ 8,    10,   12 ]      arr2 [ 7,    9,   11 ]
           ^                ^
arr3 [ 7,    8,    *,    *,    *,    * ]
           ^
```

Paso 4: Insertamos el número 10 en arr3 en su cuarta posición y avanzamos en arr1

```
arr1 [ 8,    10,   12 ]      arr2 [ 7,    9,   11 ]
           ^                ^
arr3 [ 7,    8,    9,    *,    *,    * ]
           ^
```

Paso 5: Insertamos el número 11 en arr3 en su quinta posición y avanzamos en arr2.

```
arr1 [ 8,    10,   12 ]      arr2 [ 7,    9,   11 ]
           ^                ^
arr3 [ 7,    8,    9,   10,    *,    * ]
           ^
```

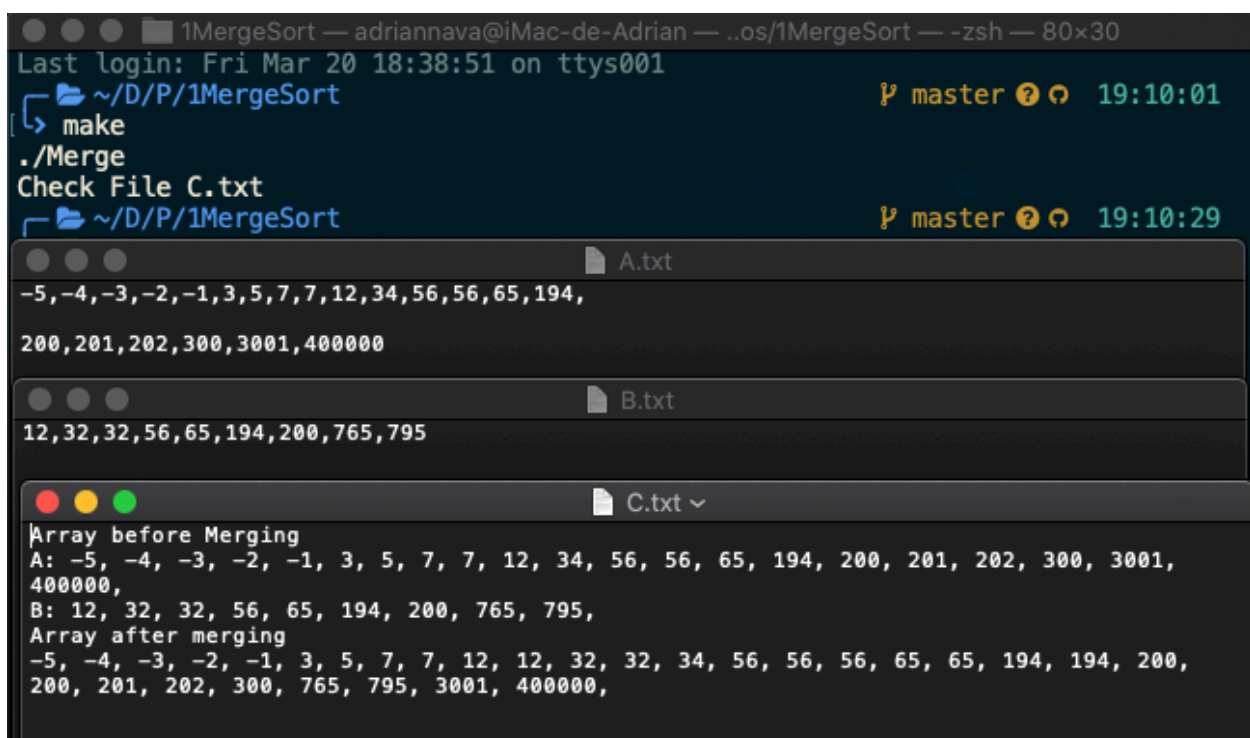
Al cumplirse el primer while donde $j = \text{arr2}$, se insertan las posiciones sobrantes de arr1 al final del arreglo.

```
arr1 [ 8,    10,   12 ]
           ^
arr3 [ 7,    8,    9,   10,   11,    *,    ]
           ^
```

Arreglo final:

```
arr3 [ 7,    8,    9,   10,   11,   12 ]
```

Capturas de Pantalla.



```
1MergeSort — adriannava@iMac-de-Adrian — .os/1MergeSort — zsh — 80x30
Last login: Fri Mar 20 18:38:51 on ttys001
~ /D/P/1MergeSort master 19:10:01
[ make
./Merge
Check File C.txt
~ /D/P/1MergeSort master 19:10:29

A.txt
-5,-4,-3,-2,-1,3,5,7,7,12,34,56,56,65,194,
200,201,202,300,3001,400000

B.txt
12,32,32,56,65,194,200,765,795

C.txt ~
Array before Merging
A: -5, -4, -3, -2, -1, 3, 5, 7, 7, 12, 34, 56, 56, 65, 194, 200, 201, 202, 300, 3001,
400000,
B: 12, 32, 32, 56, 65, 194, 200, 765, 795,
Array after merging
-5, -4, -3, -2, -1, 3, 5, 7, 7, 12, 12, 32, 32, 34, 56, 56, 56, 65, 65, 194, 194, 200,
200, 201, 202, 300, 765, 795, 3001, 400000,
```

2. Search

Pseudocódigos:

Búsqueda Binaria:

```
procedure findMaxBinary(A, value first, value last)
  if (first == last) return A[first];
  if ((last == first + 1) && numbers[first] >= numbers[last])
    return numbers[first];

  if ((last == first + 1) && numbers[first] < numbers[last])
    return numbers[last];

  first + (last - first)/2;
  if (numbers[mid] > numbers[mid + 1] && numbers[mid] > numbers[mid - 1])
    return numbers[mid];

  if (numbers[mid] > numbers[mid + 1] && numbers[mid] < numbers[mid - 1])
    return findMaxBinary(numbers, first, mid - 1);

  else return findMaxBinary(numbers, mid + 1, last);
end procedure
```

Búsqueda Linear:

```
procedure findMaxLinear(A, value first, value last)

  for each item in A
    if match item == value
      return the item's location
    end if
  end for

end procedure
```

Código Fuente:

Búsqueda Binaria

```
1.  int findMaxBinary(int numbers[], int first, int last){ //
    Complexity O(log n)
2.    if (first == last) return numbers[first];
3.    /
    * If there are two elements and first is greater then the first element
      is Max */
4.    if ((last == first + 1) && numbers[first] >= numbers[last]) return
    numbers[first];
5.    /
    * If there are two elements and second is greater then the second eleme
      nt is Max */
6.    if ((last == first + 1) && numbers[first] < numbers[last]) return
    numbers[last];
7.    int mid = (first + last)/2;    /*first + (last - first)/2;*/
```

```

8.    /
    * If we reach a point where numbers[mid] is greater than both of its ad
    jacent elements numbers[mid-1] and numbers[mid+1], then numbers[mid] is
    the Max element*/
9.    if (numbers[mid] > numbers[mid + 1] && numbers[mid] > numbers[mid -
10.    1]) return numbers[mid];
    /
    * If numbers[mid] is greater than the next element and smaller than the
    previous element then Max lies on left side of mid */
11.   if (numbers[mid] > numbers[mid + 1] && numbers[mid] < numbers[mid -
12.   1]) return findMaxBinary(numbers, first, mid - 1);
    /
    * when numbers[mid] is greater than numbers[mid-1] and smaller than num
    bers[mid+1]*/
13.   else return findMaxBinary(numbers, mid + 1, last);
14. }

```

Búsqueda Linear

```

15. int findMaxLinear(int numbers[], int first, int last){ //
    Complexity O(n)
16.   int max = numbers[first], i;
17.   /
    * break when once an element is smaller than the max then it will go on
    decreasing and no need to check after that*/
18.   for (i = first + 1; i <= last; i++){
19.       if (numbers[i] > max) max = numbers[i];
20.       else break;
21.   }
22.   return max;
23. }
24. }

```

Ejemplos:

Arreglo = [3,5,7,200,3,2,1]

Búsqueda Linear:

Encontramos el elemento más grande pasando por todos los elementos del arreglo, al final, se arroja el resultado.

Arreglo = [3, 5, 7, 200, 3, 2, 1]

^

max = 3

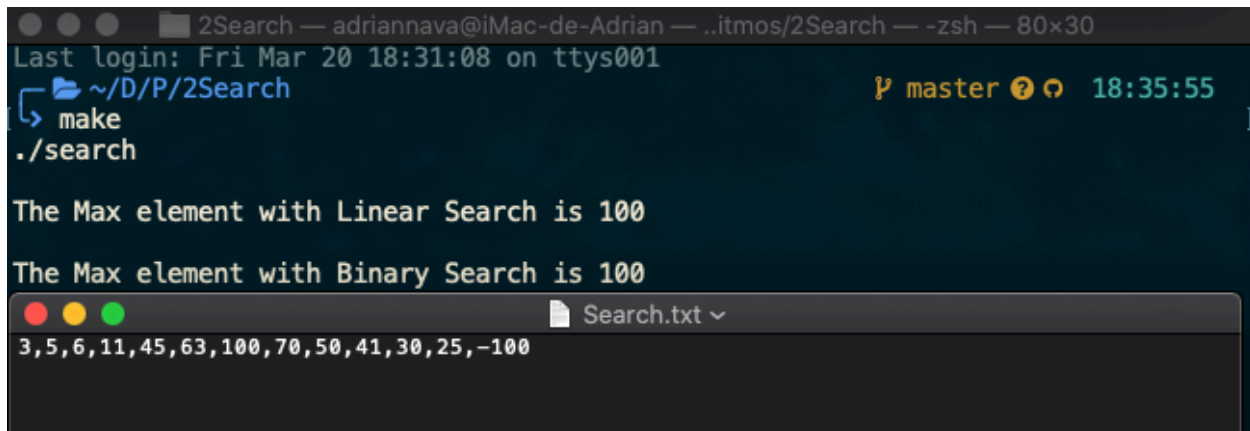
Arreglo = [3, 5, 7, 200, 3, 2, 1]

^

max = 5

Max = 200

Capturas de Pantalla.



```
2Search — adriannava@iMac-de-Adrian — ..itmos/2Search — -zsh — 80x30
Last login: Fri Mar 20 18:31:08 on ttys001
~ /D/P/2Search
[ make
./search ]
P master 18:35:55

The Max element with Linear Search is 100

The Max element with Binary Search is 100

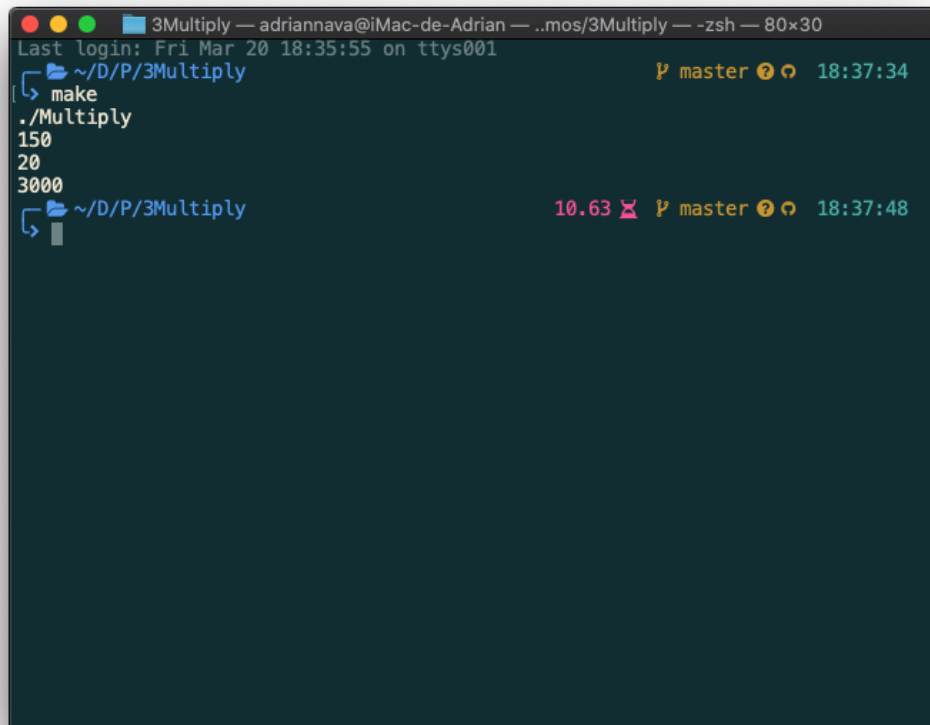
Search.txt
3,5,6,11,45,63,100,70,50,41,30,25,-100
```

3. Multiplicación.

Código Fuente.

```
1. int multiply(int x, int y){
2.     int z;
3.     if (y == 0) return 0;
4.     z = multiply(x,abs(y/2));
5.     if (y % 2 == 0) return 2 * z;
6.     else return x + (2 * z);
7. }
```

Captura de Pantalla.



```
3Multiply — adriannava@iMac-de-Adrian — ..mos/3Multiply — -zsh — 80x30
Last login: Fri Mar 20 18:35:55 on ttys001
~ /D/P/3Multiply
[ make
./Multiply ]
P master 18:37:34
150
20
3000
~ /D/P/3Multiply
10.63 P master 18:37:48
```

4. Coordenadas.

Pseudocódigo.

```
procedure ClosestPoints
    minDist = infinity
    for i = 1 to length(P) - 1 do          O(n)
        for j = i + 1 to length(P) do      O(n*n)
            let p = P[i], q = P[j]
            if dist(p, q) < minDist then
                minDist = dist(p, q)
                closestPair = (p, q)
    return closestPair
end procedure
```

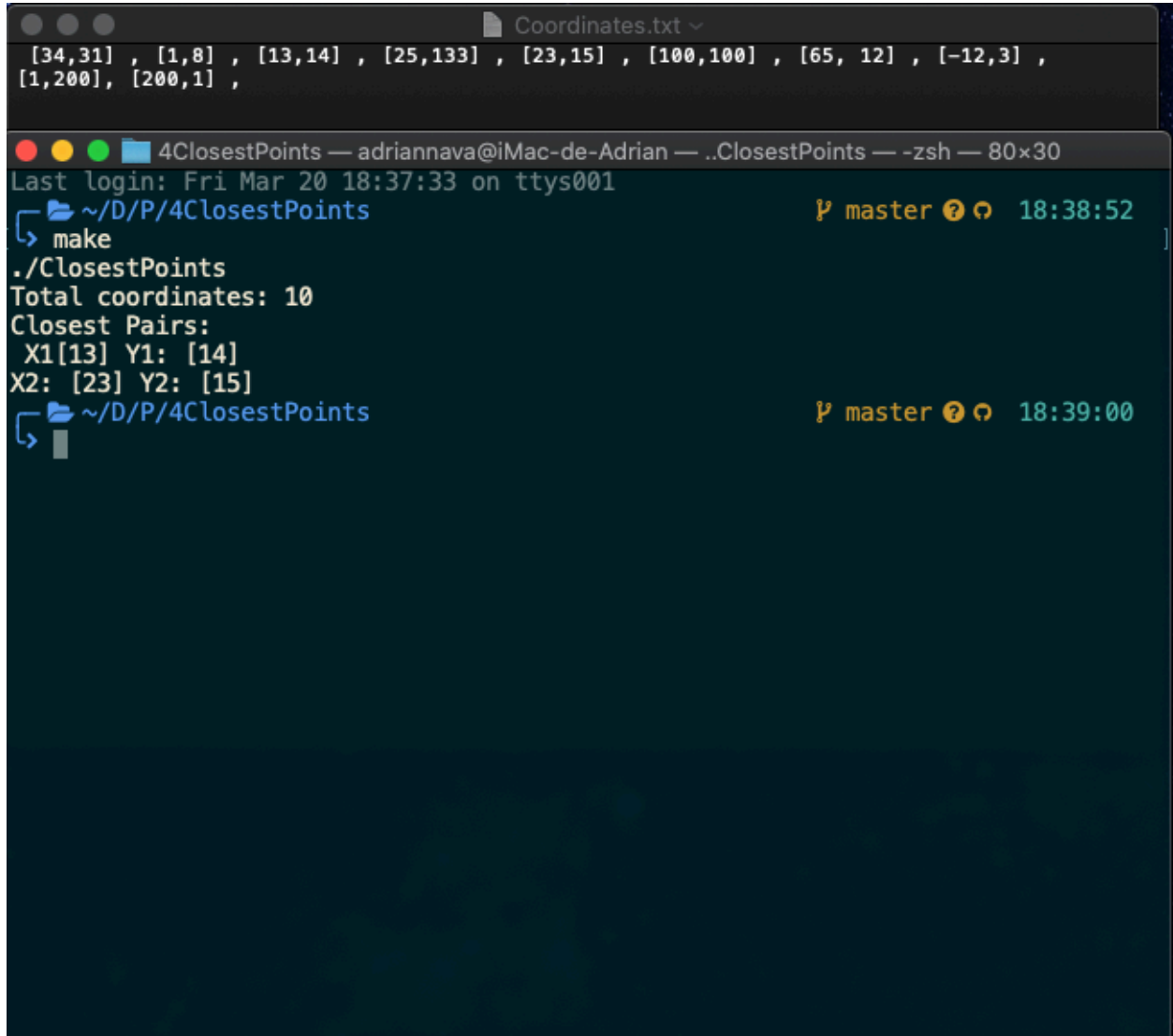
Complejidad.

$O(n*n) \rightarrow O(n^2)$

Código Fuente.

```
1. struct Point{
2.     int x, y;
3. };
4.
5. //Complexity O(n^2)
6. struct Point * ClosestPoints(struct Point P[], int n){
7.     float currClo = FLT_MAX, dist = 0.0;
8.     int i, j;
9.     if ((n < 2) || (P == NULL)){
10.         fprintf(stderr, "Error: not enough points to find closest pair");
11.         exit(EXIT_FAILURE);
12.     }
13.     struct Point * closestPair = (struct Point *)malloc(2 * (sizeof(struct
14.         t Point)));
15.     if (closestPair == NULL){
16.         fprintf(stderr, "Error: memory allocation failed");
17.         exit(EXIT_FAILURE);
18.     }
19.     for (i = 0; i < n; ++i){
20.         for (j = i + 1; j < n; ++j){ //sqrt (px - qx)^2 + (py - qy)^2
21.             dist = sqrt( (P[i].x - P[j].x)*(P[i].x - P[j].x) +
22.                 (P[i].y - P[j].y)*(P[i].y - P[j].y));
23.             if (dist < currClo){
24.                 currClo = dist;
25.                 closestPair[0] = P[i];
26.                 closestPair[1] = P[j];
27.             }
28.         }
29.     }
30.     return closestPair;
31. }
```


Capturas de Pantalla



```
Coordinates.txt
[34,31] , [1,8] , [13,14] , [25,133] , [23,15] , [100,100] , [65, 12] , [-12,3] ,
[1,200], [200,1] ,

4ClosestPoints — adriannava@iMac-de-Adrian — ../ClosestPoints — zsh — 80x30
Last login: Fri Mar 20 18:37:33 on ttys001
~/D/P/4ClosestPoints master 18:38:52
└─ make
./ClosestPoints
Total coordinates: 10
Closest Pairs:
X1[13] Y1: [14]
X2: [23] Y2: [15]
~/D/P/4ClosestPoints master 18:39:00
└─
```

The screenshot shows a macOS terminal window with a dark background. The top window, titled 'Coordinates.txt', displays a list of 10 coordinate pairs: [34,31], [1,8], [13,14], [25,133], [23,15], [100,100], [65, 12], [-12,3], [1,200], and [200,1]. Below it, a terminal window titled '4ClosestPoints' shows the execution of a program. The terminal output indicates that there are 10 total coordinates and identifies the closest pair as points at (13, 14) and (23, 15). The terminal window also shows the current directory as ~/D/P/4ClosestPoints and the status of the 'master' branch.