

**Instituto**

**Politécnico**

**Nacional**

**Escuela Superior de Cómputo**

**REPORTE DE LA SESIÓN 4: Divide y vencerás 1**

**Materia:**

Análisis de algoritmos

**Grupo:**

3CM3

**Integrantes:**

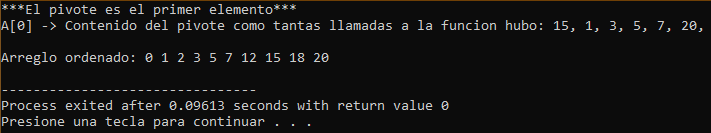
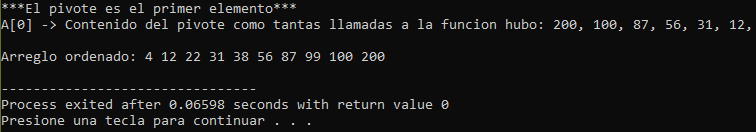
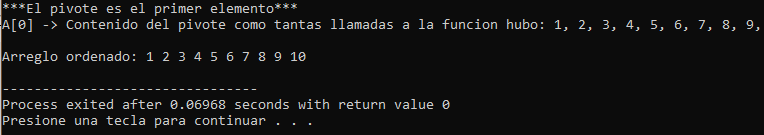
Castro Cruces Jorge Eduardo

**Fecha:**

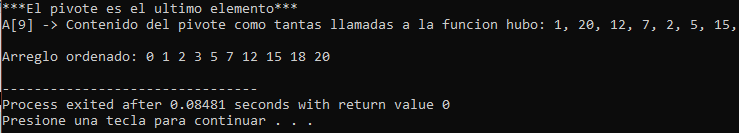
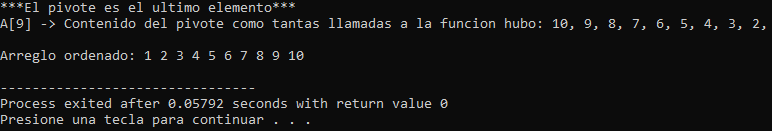
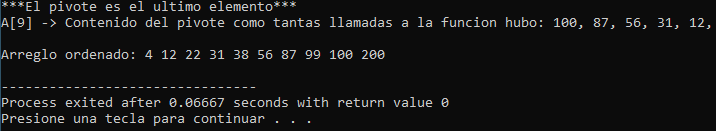
Miércoles, abril 22, 2020Ejercicio 1

Implementa el algoritmo Quicksort, considera cada uno de los siguientes casos:

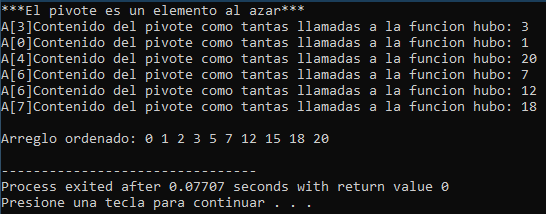
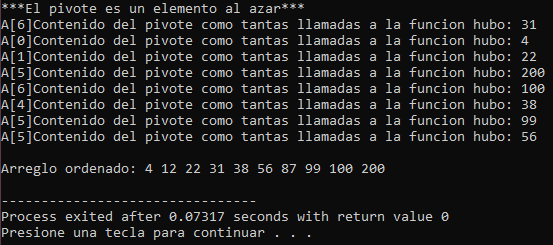
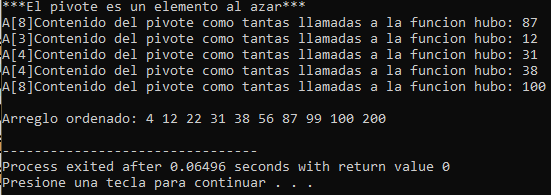
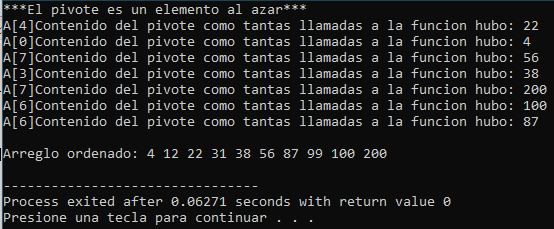
1. Elige siempre el primer elemento como pivote
2. #include<stdio.h>
4. **void** Intercambiar(**int**\* a, **int**\* b) {
5. **int** t = \*a;
6. \*a = \*b;
7. \*b = t;
8. }
10. **int** Part(**int** A[], **int** Menor, **int** Mayor){
11. **int** Piv = A[Menor];
12. **int** i = Menor, j;
13. printf("%d, ", Piv);
14. **for**(j=Menor+1; j<=Mayor; j++){
15. **if**(Piv>=A[j]){
16. i++;
17. Intercambiar(&A[i], &A[j]);
18. }
19. }
20. Intercambiar(&A[i], &A[Menor]);
21. **return** i;
22. }
24. **void** QuickSort(**int** A[], **int** Menor, **int** Mayor){
25. **if**(Menor < Mayor){
26. **int** Piv = Part(A, Menor, Mayor);
27. QuickSort(A, Menor, Piv-1);
28. QuickSort(A, Piv+1, Mayor);
29. }
30. }
32. **void** ImprimirA(**int** A[], **int** n){
33. **int** i;
34. **for** (i=0; i<n; i++)
35. printf("%d ", A[i]);
36. printf("\n");
37. }
39. **int** main(){
40. **int** A[] = {15, 20, 5, 3, 2, 0, 7, 18, 12, 1};
41. **int** n = **sizeof**(A)/**sizeof**(A[0]);
42. printf("\*\*\*El pivote es el primer elemento\*\*\*\n");
43. printf("A[0] -> Contenido del pivote como tantas llamadas a la funcion hubo: ");
44. QuickSort(A, 0, n-1);
45. printf("\n\nArreglo ordenado: ");
46. ImprimirA(A, n);
47. **return** 0;
48. }

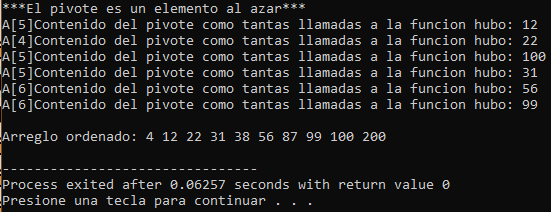
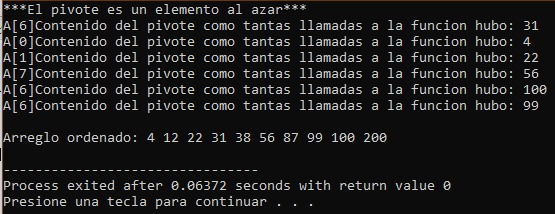
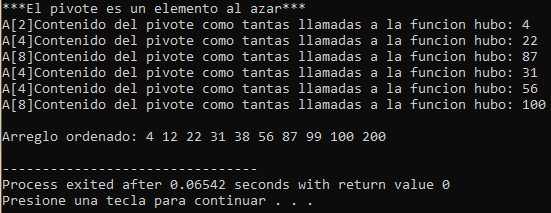
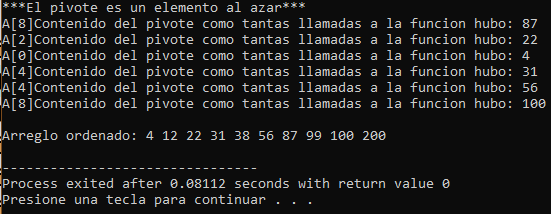


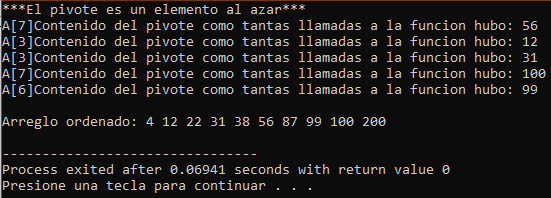
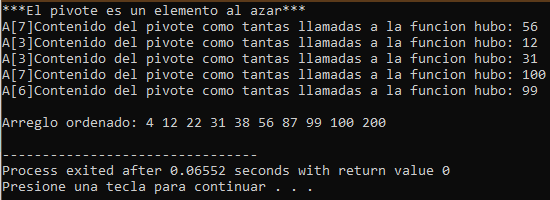
1. Elige siempre el último elemento como pivote
2. #include<stdio.h>
4. **void** Intercambiar(**int**\* a, **int**\* b) {
5. **int** t = \*a;
6. \*a = \*b;
7. \*b = t;
8. }
10. **int** Part(**int** A[], **int** Menor, **int** Mayor){
11. **int** Piv = A[Mayor];
12. **int** i = (Menor - 1), j;
13. printf("%d, ", Piv);
14. **for**(j=Menor; j<=Mayor-1; j++){
15. **if** (A[j] < Piv){
16. i++;
17. Intercambiar(&A[i], &A[j]);
18. }
19. }
20. Intercambiar(&A[i+1], &A[Mayor]);
21. **return** (i+1);
22. }
24. **void** QuickSort(**int** A[], **int** Menor, **int** Mayor){
25. **if**(Menor < Mayor){
26. **int** Piv = Part(A, Menor, Mayor);
27. QuickSort(A, Menor, Piv-1);
28. QuickSort(A, Piv+1, Mayor);
29. }
30. }
32. **void** ImprimirA(**int** A[], **int** n){
33. **int** i;
34. **for** (i=0; i<n; i++)
35. printf("%d ", A[i]);
36. printf("\n");
37. }
39. **int** main(){
40. **int** A[] = {15, 20, 5, 3, 2, 0, 7, 18, 12, 1};
41. **int** n = **sizeof**(A)/**sizeof**(A[0]);
42. printf("\*\*\*El pivote es el ultimo elemento\*\*\*\n");
43. printf("A[%d] -> Contenido del pivote como tantas llamadas a la funcion hubo: ", n-1);
44. QuickSort(A, 0, n-1);
45. printf("\n\nArreglo ordenado: ");
46. ImprimirA(A, n);
47. **return** 0;
48. }



1. Elige al azar el elemento pivote.
2. #include<stdio.h>
4. **void** Intercambiar(**int**\* a, **int**\* b) {
5. **int** t = \*a;
6. \*a = \*b;
7. \*b = t;
8. }
10. **int** Part(**int** A[], **int** Menor, **int** Mayor, **int** Rand){
11. **int** Piv = A[Mayor];
12. **int** i = (Menor-1), j;
13. printf("A[%d]Contenido del pivote como tantas llamadas a la funcion hubo: ", Rand);
14. printf("%d\n", Piv);
15. **for**(j=Menor; j<=(Mayor-1); j++){
16. **if**(A[j]<=Piv){
17. i++;
18. Intercambiar(&A[i], &A[j]);
19. }
20. }
21. Intercambiar(&A[i+1], &A[Mayor]);
22. **return** (i+1);
23. }
25. **int** PartR(**int** A[], **int** Menor, **int** Mayor){
26. srand(time(NULL));
27. **int** Rand = Menor + rand() % (Mayor-Menor);
28. Intercambiar(&A[Rand], &A[Mayor]);
29. **return** Part(A, Menor, Mayor, Rand);
30. }
32. **void** QuickSort(**int** A[], **int** Menor, **int** Mayor){
33. **if**(Menor < Mayor){
34. **int** Piv = PartR(A, Menor, Mayor);
35. QuickSort(A, Menor, Piv-1);
36. QuickSort(A, Piv+1, Mayor);
37. }
38. }
40. **void** ImprimirA(**int** A[], **int** n){
41. **int** i;
42. **for** (i=0; i<n; i++)
43. printf("%d ", A[i]);
44. printf("\n");
45. }
47. **int** main(){
48. **int** A[] = {15, 20, 5, 3, 2, 0, 7, 18, 12, 1};
49. **int** n = **sizeof**(A)/**sizeof**(A[0]);
50. printf("\*\*\*El pivote es un elemento al azar\*\*\*\n");
51. QuickSort(A, 0, n-1);
52. printf("\nArreglo ordenado: ");
53. ImprimirA(A, n);
54. **return** 0;
55. }







Calculando el promedio de los 10 resultados anteriores, se obtuvo el siguiente resultado:

**0.068567 segundos de tiempo de ejecución**

**6 comparaciones para cada ejecución**

Conclusiones

Una vez concluida esta práctica, podemos decir que los objetivos se cumplieron al llevar a cabo el análisis, implementación y ejecución de las tres distintas formas de elegir un pivote para el algoritmo de QuickSort.

Comparando los tres distintos métodos, podemos concluir que tercero es el más eficiente, ya que al ser escogido un pivote aleatorio, mejoramos la complejidad de tiempo esperada o promedio para **O (N log N).**La complejidad del peor caso sigue siendo **O (N ^ 2)**.