

# PROGRAMA PROFESIONAL

Ciencias de la Computación

# **TÍTULO DEL TRABAJO**

Comparación de Algoritmos

# **CURSO**

Análisis y Diseño de Algoritmos

#### **ALUMNOS**

• Angel Josue Loayza Huarachi

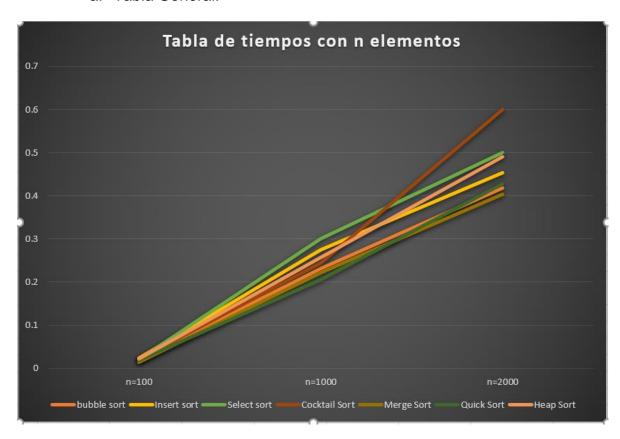
SEMESTRE: V

**AÑO**: 2022

"El alumno declara haber realizado el presente trabajo de acuerdo a las normas de la Universidad Católica San Pablo

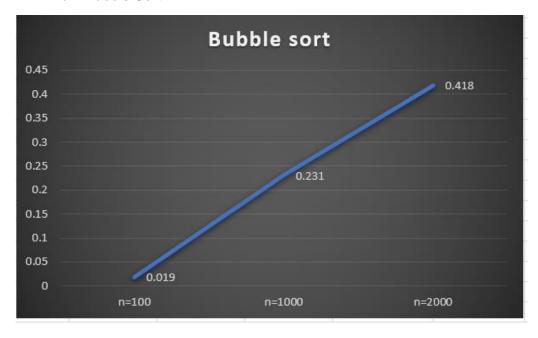
### 1. Gráficos

### a. Tabla General:



	bubble sort	Insert sort	Select sort	Cocktail Sort	Merge Sort	Quick Sort	Heap Sort
n=100	0.019	0.023	0.02	0.02	0.014	0.019	0.022
n=1000	0.231	0.275	0.3	0.244	0.222	0.205	0.257
n=2000	0.418	0.454	0.5	0.6	0.404	0.427	0.491

### b. Bubble Sort



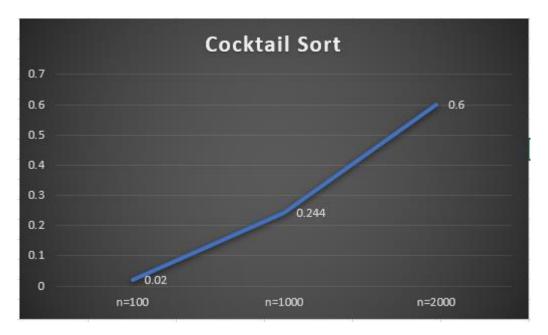
### c. Insert Sort



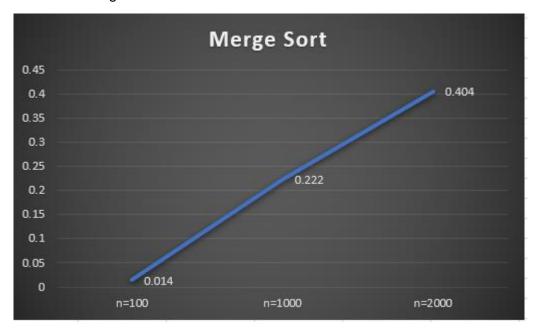
d. Select Sort



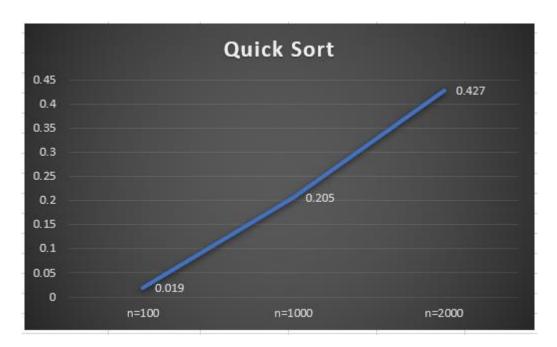
e. Cocktail Sort



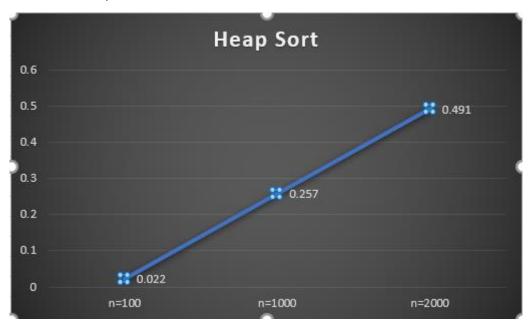
f. Merge Sort



g. Quick Sort



h. Heap Sort



```
2. Códigos
3. #include <iostream>
4. #include <ctime>
5. //#include "Header.h"
using namespace std;
7.
8. void printArray(int A[], int size)
9. {
10.
       cout << "-----
                                        -----" << endl;
       for (auto i = 0; i < size; i++)
    cout << A[i] << " ";</pre>
11.
12.
13.}
14.
15.//bouble random
16.void boobler(int a[], int n)
17. {
18.
       printArray(a, n);
19.
       cout << endl;</pre>
20.
21.
       srand(time(NULL));
22.
       int temp;
23.
24.
       for (int i = 0; i < n; i++) {</pre>
25.
            for (int j = i + 1; j < n; j++)
26.
27.
                if (a[j] < a[i]) {</pre>
28.
                    temp = a[i];
29.
                    a[i] = a[j];
30.
                    a[j] = temp;
31.
                }
32.
            }
33.
       }
34.
       printArray(a, n);
35.}
36.
37.//insert random
38.void insertr(int a[], int n)
39. {
40.
       printArray(a, n);
41.
       cout << endl;</pre>
42.
43.
       srand(time(NULL));
44.
45.
       for (int i = 1; i < n; i++)</pre>
46.
47.
            int temp = a[i];
48.
            int j = i - 1;
49.
            while (j >= 0 && temp <= a[j])</pre>
50.
51.
                a[j + 1] = a[j];
                j = j - 1;
52.
53.
            a[j + 1] = temp;
54.
55.
57.
       printArray(a, n);
58.}
59.
60.//select random
61.void selectionr(int a[], int n)
62. {
63.
       printArray(a, n);
64.
       cout << endl;</pre>
```

```
65.
       srand(time(NULL));
66.
67.
68.
       int i, j, loc, temp, min;
69.
       for (i = 0; i < n - 1; i++)
70.
71.
72.
            min = a[i];
73.
            loc = i;
74.
75.
            for (j = i + 1; j < n; j++)
76.
77.
                if (min > a[j])
78.
79.
                     min = a[j];
80.
                    loc = j;
81.
                }
82.
            }
83.
            temp = a[i];
84.
            a[i] = a[loc];
85.
86.
            a[loc] = temp;
87.
88.
89.
       printArray(a, n);
90.}
91.
92.//cocktail random
93. void cocktailr(int a[], int n)
94. {
95.
       printArray(a, n);
       cout << endl;</pre>
96.
97.
       srand(time(NULL));
100.
              bool swapped = true;
101.
              int start = 0;
102.
              int end = n - 1;
103.
104.
              while (swapped) {
105.
                   swapped = false;
106.
107.
                   for (int i = start; i < end; ++i) {</pre>
108.
                       if (a[i] > a[i + 1]) {
109.
                           swap(a[i], a[i + 1]);
110.
                           swapped = true;
                       }
111.
                   }
112.
113.
114.
                   if (!swapped)
115.
                       break;
116.
                   swapped = false;
117.
118.
                   --end;
                   for (int i = end - 1; i >= start; --i) {
119.
                       if (a[i] > a[i + 1]) {
120.
                           swap(a[i], a[i + 1]);
121.
122.
                           swapped = true;
                       }
123.
124.
                   }
125.
                   ++start;
126.
              }
127.
```

```
128.
              printArray(a, n);
129.
130.
131.
          //merge random
          void mergeauxr(int array[], int const left, int const mid, int
132.
   const right)
133.
          {
              auto const subArrayOne = mid - left + 1;
134.
135.
              auto const subArrayTwo = right - mid;
136.
137.
              // Create temp arrays
138.
              auto* leftArray = new int[subArrayOne],
139.
                  * rightArray = new int[subArrayTwo];
140
141.
              // Copy data to temp arrays leftArray[] and rightArray[]
142.
              for (auto i = 0; i < subArrayOne; i++)</pre>
143.
                  leftArray[i] = array[left + i];
144.
              for (auto j = 0; j < subArrayTwo; j++)</pre>
145.
                  rightArray[j] = array[mid + 1 + j];
146.
              auto indexOfSubArrayOne = 0, // Initial index of first sub-
147.
   arrav
                  indexOfSubArrayTwo = 0; // Initial index of second sub-
148.
   array
              int indexOfMergedArray = left; // Initial index of merged
149.
   array
150.
              // Merge the temp arrays back into array[left..right]
151.
              while (indexOfSubArrayOne < subArrayOne &&</pre>
   indexOfSubArrayTwo < subArrayTwo) {</pre>
152.
                  if (leftArray[indexOfSubArrayOne] <=</pre>
   rightArray[indexOfSubArrayTwo]) {
153.
                      array[indexOfMergedArray] =
   leftArray[indexOfSubArrayOne];
154.
                      indexOfSubArrayOne++;
155.
                  }
156.
                  else {
                      array[indexOfMergedArray] =
   rightArray[indexOfSubArrayTwo];
158.
                      indexOfSubArrayTwo++;
159.
160.
                  indexOfMergedArray++;
161.
              // Copy the remaining elements of
162.
163.
              // left[], if there are any
164.
              while (indexOfSubArrayOne < subArrayOne) {</pre>
165.
                  array[indexOfMergedArray] =
   leftArray[indexOfSubArrayOne];
166.
                  indexOfSubArrayOne++;
                  indexOfMergedArray++;
167.
168.
              // Copy the remaining elements of
169.
170.
              // right[], if there are any
171.
              while (indexOfSubArrayTwo < subArrayTwo) {</pre>
172.
                  array[indexOfMergedArray] =
   rightArray[indexOfSubArrayTwo];
173.
                  indexOfSubArrayTwo++;
174.
                  indexOfMergedArray++;
              }
175.
          }
176.
177.
          void merger(int array[], int const begin, int const end)
178.
179.
              if (begin >= end)
180.
```

```
return; // Returns recursively
181.
182.
              auto mid = begin + (end - begin) / 2;
183.
              merger(array, begin, mid);
184.
185.
              merger(array, mid + 1, end);
186.
              mergeauxr(array, begin, mid, end);
          }
187.
188.
          //quick random
189.
190.
          int partitionr(int arr[], int start, int end)
191.
192.
              int pivot = arr[start];
193.
194.
195.
              int count = 0;
196.
              for (int i = start + 1; i <= end; i++) {</pre>
197.
                  if (arr[i] <= pivot)</pre>
                       count++;
198.
199.
              }
200.
201.
              // Giving pivot element its correct position
202.
              int pivotIndex = start + count;
203.
              swap(arr[pivotIndex], arr[start]);
204.
              // Sorting left and right parts of the pivot element
205.
              int i = start, j = end;
206.
207.
              while (i < pivotIndex && j > pivotIndex) {
208.
209.
210.
                  while (arr[i] <= pivot) {</pre>
211.
                       i++;
212.
213.
                  while (arr[j] > pivot) {
214.
215.
                       j--;
216.
217.
218.
                  if (i < pivotIndex && j > pivotIndex) {
219.
                       swap(arr[i++], arr[j--]);
                  }
220.
221.
              }
222.
223.
              return pivotIndex;
224.
          }
225.
226.
          void quickr(int arr[], int start, int end)
227.
228.
229.
              // base case
              if (start >= end)
230.
231.
                  return;
232.
233.
              // partitioning the array
234.
              int p = partitionr(arr, start, end);
235.
              // Sorting the left part
236.
              quickr(arr, start, p - 1);
237.
238.
239.
              // Sorting the right part
              quickr(arr, p + 1, end);
240.
241.
          }
242.
243.
          //heap random
```

```
void heapifyr(int arr[], int n, int i)
244.
245.
              int largest = i; // Initialize largest as root
246.
247.
              int l = 2 * i + 1; // left = 2*i + 1
248.
             int r = 2 * i + 2; // right = 2*i + 2
249.
250.
              // If left child is larger than root
251.
              if (l < n && arr[l] > arr[largest])
252.
                  largest = l;
253.
254.
              // If right child is larger than largest so far
255.
              if (r < n && arr[r] > arr[largest])
256.
                  largest = r;
257.
              // If largest is not root
258.
259.
              if (largest != i) {
260.
                  swap(arr[i], arr[largest]);
261.
262.
                  // Recursively heapify the affected sub-tree
263.
                  heapifyr(arr, n, largest);
              }
264.
265.
         }
266.
         // main function to do heap sort
267.
         void heapr(int arr[], int n)
268.
269.
              // Build heap (rearrange array)
270.
              for (int i = n / 2 - 1; i \ge 0; i--)
271.
                  heapifyr(arr, n, i);
272.
273.
274.
              // One by one extract an element from heap
              for (int i = n - 1; i \ge 0; i--) {
275.
276.
                  // Move current root to end
277.
                  swap(arr[0], arr[i]);
278.
279.
                  // call max heapify on the reduced heap
280.
                  heapifyr(arr, i, 0);
281.
              }
282.
         }
283.
284.
         int main()
285.
              // ----- Creacion de numeros aleatorios -----
286.
287.
              int a[2000];
288.
              int n = 2000;
289.
290.
              for (int i = 0; i < n; i++)</pre>
291.
                  a[i] = rand() % (100 - 1 + 1) + 1;
292.
293.
              //---- Algoritmos ------
294.
295.
              unsigned t0, t1;
296.
              t0 = clock();
297.
              //boobler(a,n);
298.
299.
              //insertr(a,n);
              //selectionr(a,n);
300.
              //cocktailr(a,n);
301.
302.
             /*
303.
             printArray(a, n);
304.
              cout << endl;</pre>
305.
306.
              merger(a, 0, n - 1);
```

```
307.
                printArray(a, n);
308.
                */
309.
310.
                printArray(a, n);
311.
                cout << endl;
312.
313.
                quickr(a, 0, n - 1);
                printArray(a, n);
314.
315.
                */
316.
317.
                printArray(a, n);
                cout << endl;</pre>
318.
319.
                heapr(a, n);
320.
                printArray(a, n);
321.
322.
323.
                t1 = clock();
324.
                double time = (double(t1 - t0) / CLOCKS_PER_SEC);
cout << endl << "Execution Time (s): " << time << endl;</pre>
325.
326.
327.
328.
                return 0;
329.
           }
```