Universidad Católica
San Pablo

PROGRAMA PROFESIONAL

Ciencias de la Computación

TÍTULO DEL TRABAJO

Comparación de Algoritmos

CURSO

Análisis y Diseño de Algoritmos

ALUMNOS

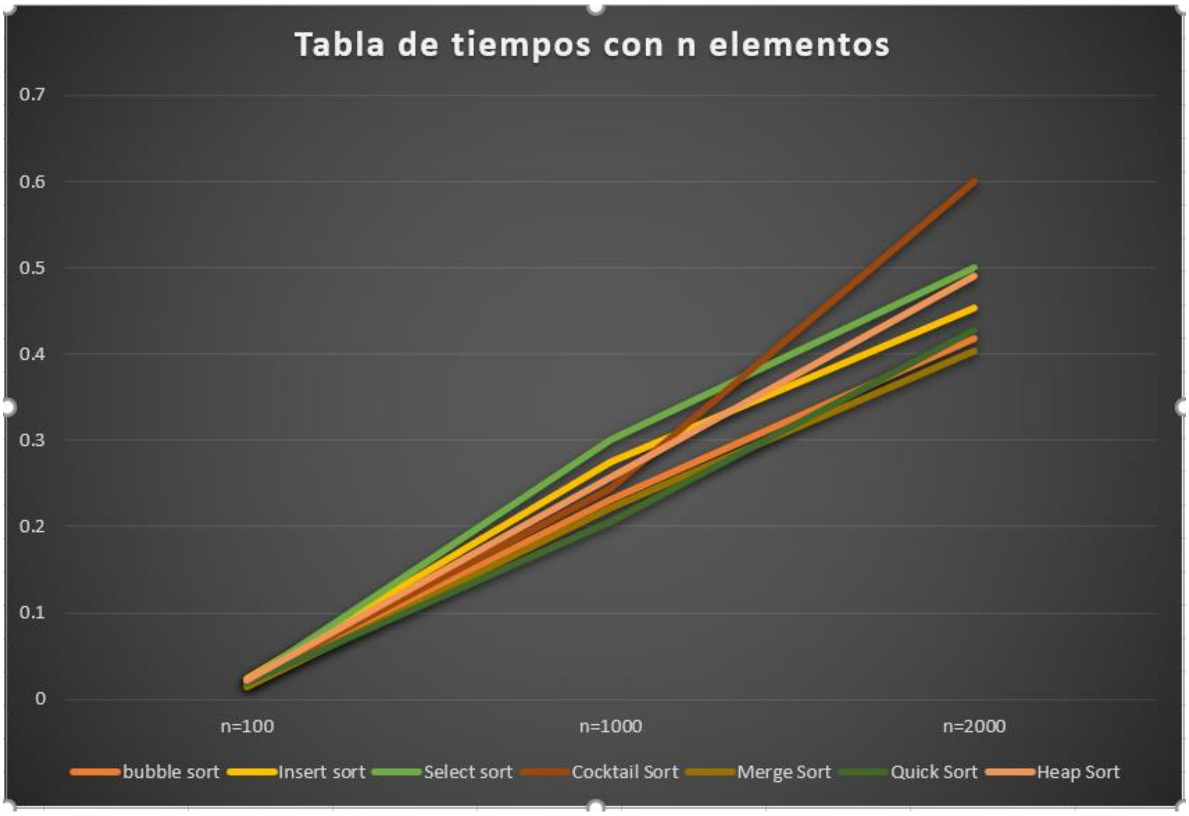
- Angel Josue Loayza Huarachi

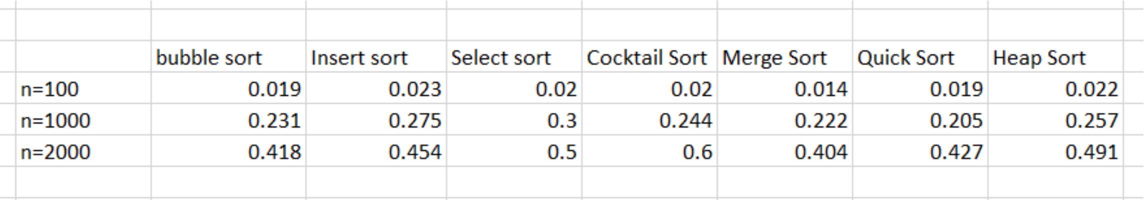
SEMESTRE: V

AÑO: 2022

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

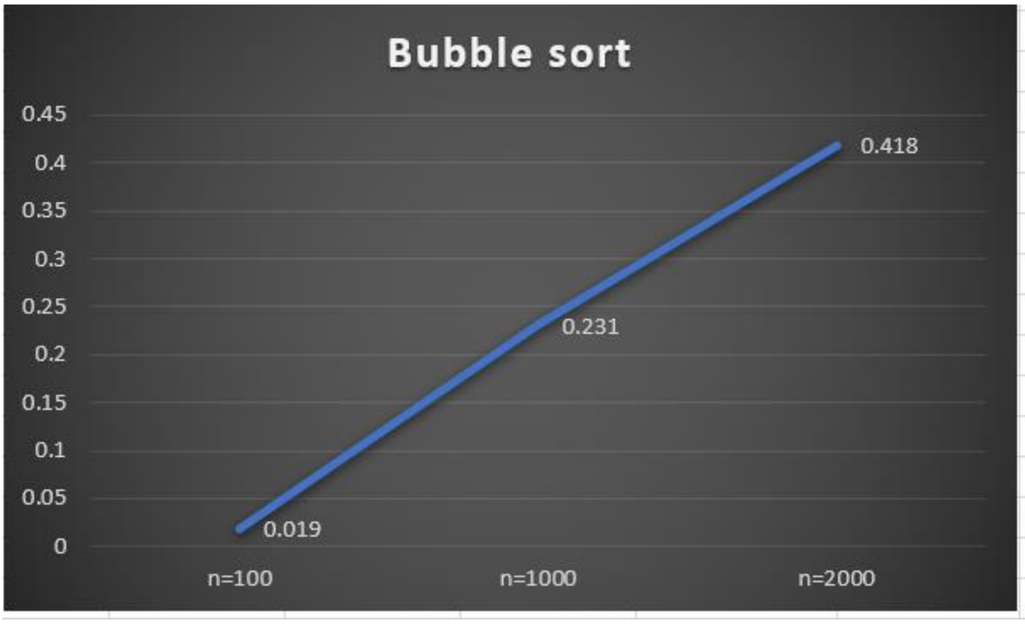
1. Gráficos
    a. Tabla General:



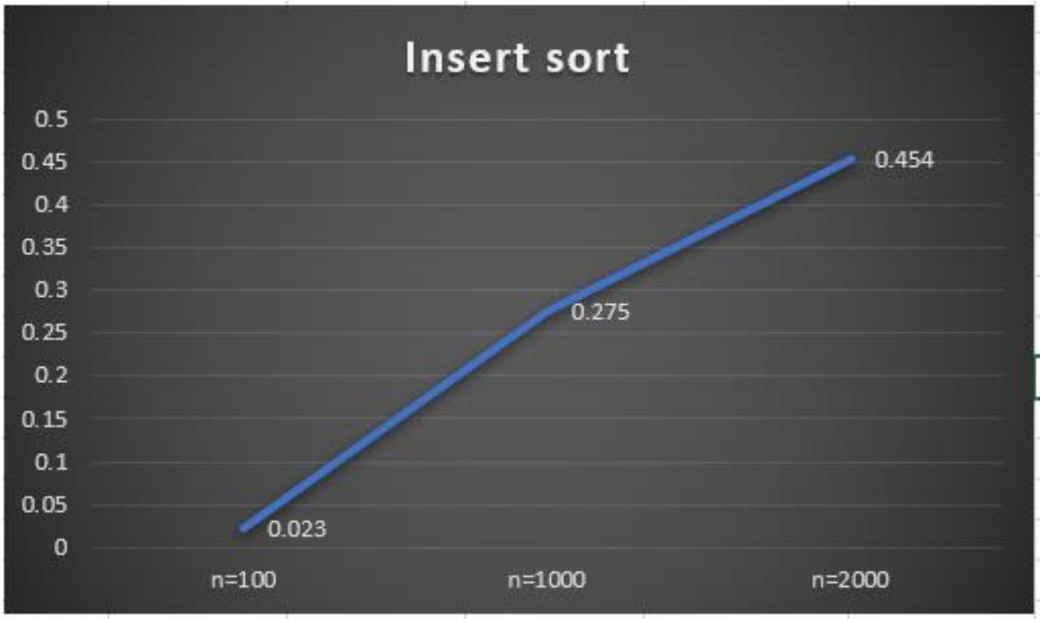


	bubble sort	Insert sort	Select sort	Cocktail Sort	Merge Sort	Quick Sort	Heap Sort
n=100	0.019	0.023	0.02	0.02	0.014	0.019	0.022
n=1000	0.231	0.275	0.3	0.244	0.222	0.205	0.257
n=2000	0.418	0.454	0.5	0.6	0.404	0.427	0.491

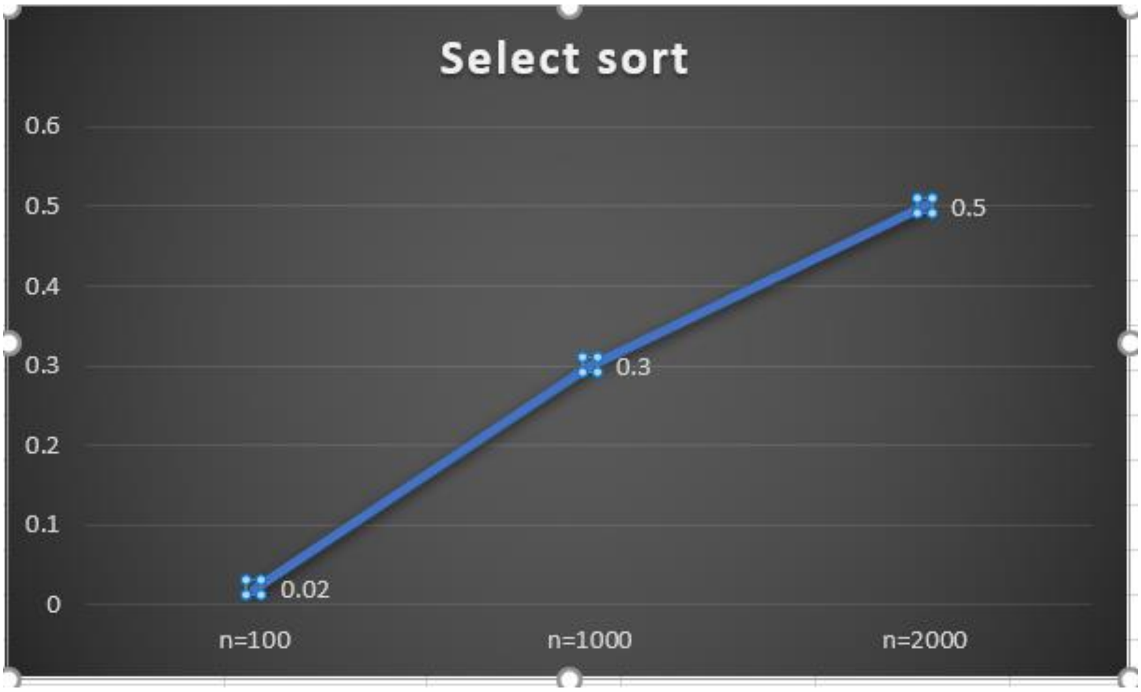
    b. Bubble Sort



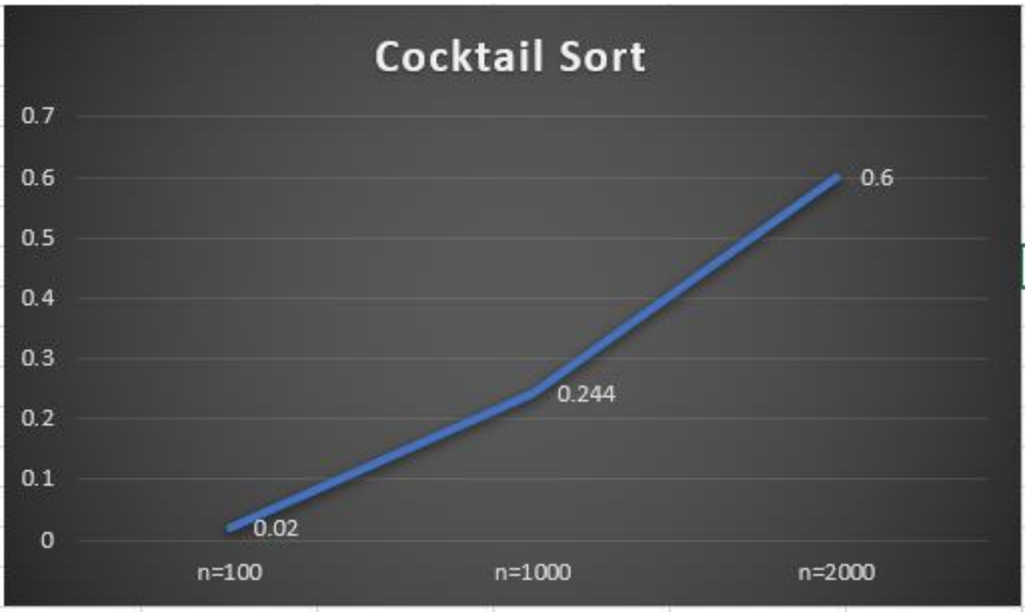
c. Insert Sort



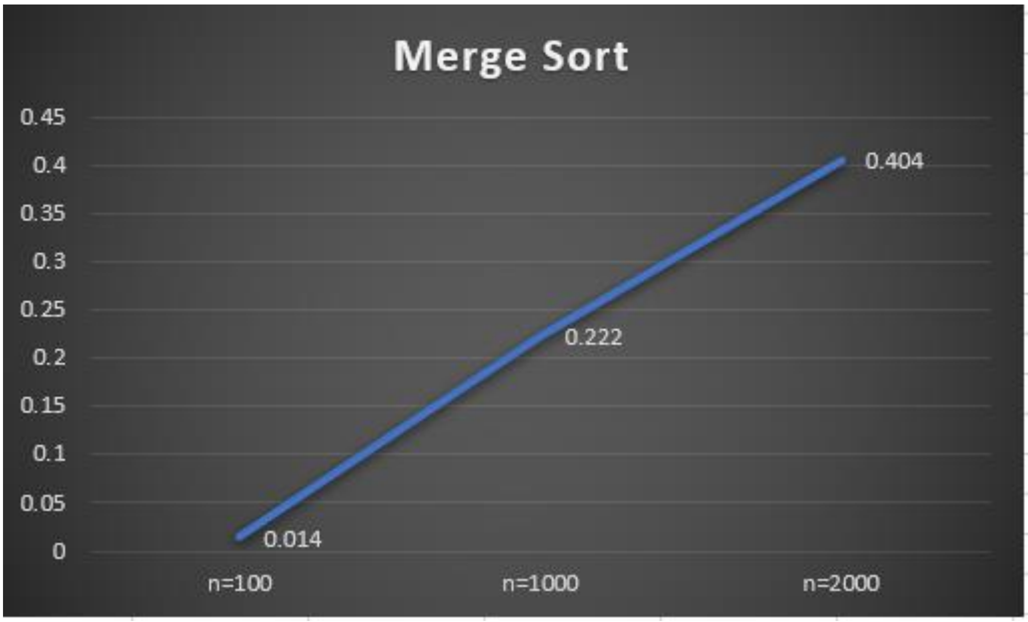
d. Select Sort



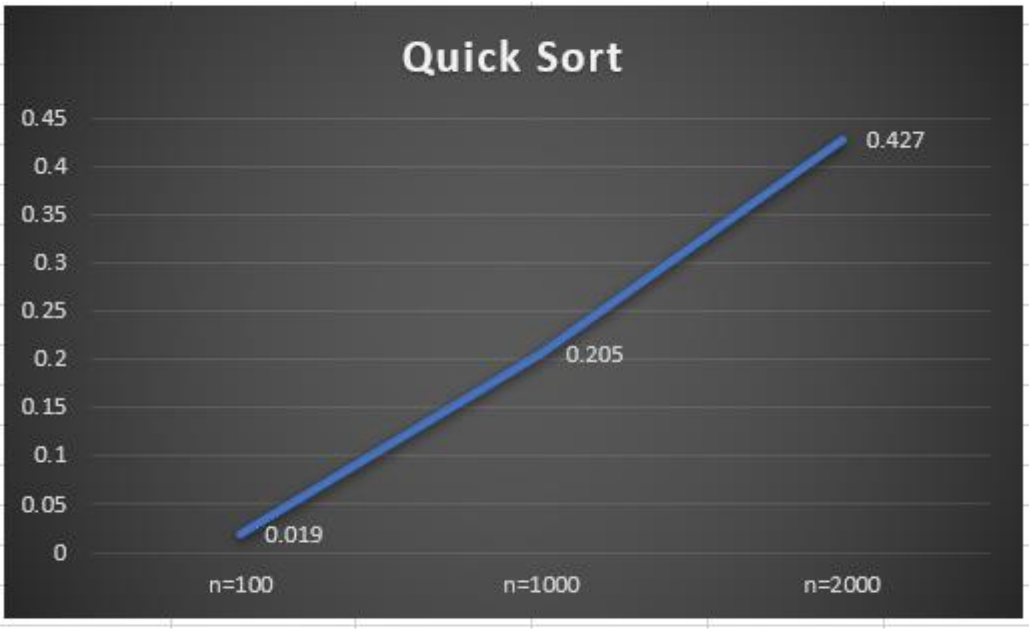
e. Cocktail Sort



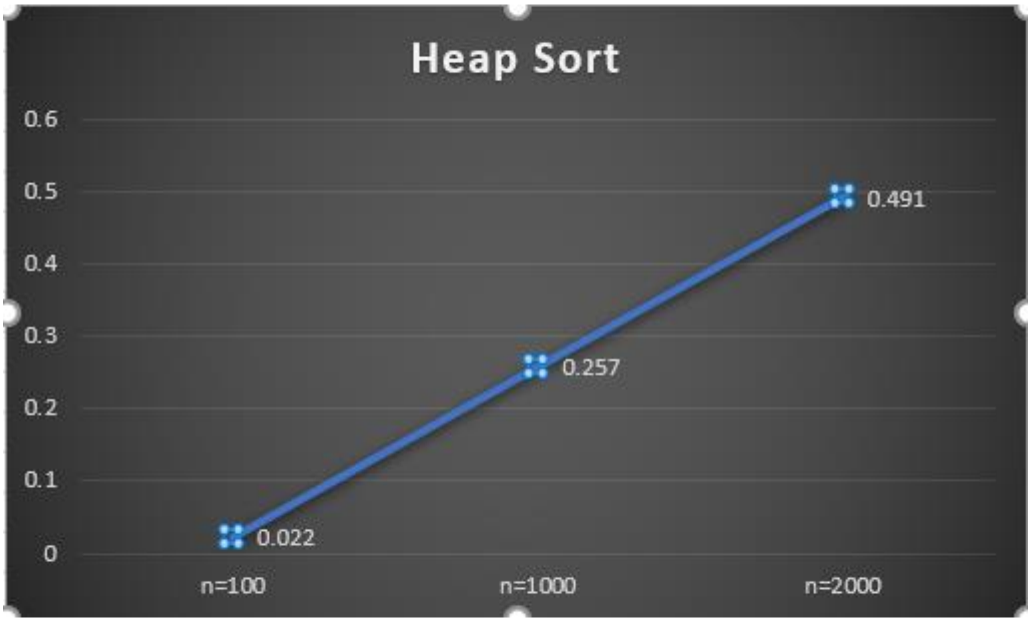
f. Merge Sort



g. Quick Sort


Quick Sort
0.45
0.4
0.35
0.3
0.25
0.2
0.15
0.1
0.05
0
0.019
0.205
0.427
n=100
n=1000
n=2000

h. Heap Sort


Heap Sort
0.6
0.5
0.4
0.3
0.2
0.1
0
0.022
0.257
0.491
n=100
n=1000
n=2000

2. Códigos

```cpp
#include <iostream>
#include <ctime>
//#include "Header.h"
using namespace std;

void printArray(int A[], int size)
{
    cout << "-----------------------------------------------" << endl;
    for (auto i = 0; i < size; i++)
        cout << A[i] << " ";
}

//bouble random
void boobler(int a[], int n)
{
    printArray(a, n);
    cout << endl;

    srand(time(NULL));
    int temp;

    for (int i = 0; i < n; i++) {
        for (int j = i + 1; j < n; j++)
        {
            if (a[j] < a[i]) {
                temp = a[i];
                a[i] = a[j];
                a[j] = temp;
            }
        }
    }
    printArray(a, n);
}

//insert random
void insertr(int a[], int n)
{
    printArray(a, n);
    cout << endl;

    srand(time(NULL));

    for (int i = 1; i < n; i++)
    {
        int temp = a[i];
        int j = i - 1;
        while (j >= 0 && temp <= a[j])
        {
            a[j + 1] = a[j];
            j = j - 1;
        }
        a[j + 1] = temp;
    }

    printArray(a, n);
}

//select random
void selectionr(int a[], int n)
{
    printArray(a, n);
    cout << endl;

    srand(time(NULL));

    int i, j, loc, temp, min;

    for (i = 0; i < n - 1; i++)
    {
        min = a[i];
        loc = i;

        for (j = i + 1; j < n; j++)
        {
            if (min > a[j])
            {
                min = a[j];
                loc = j;
            }
        }

        temp = a[i];
        a[i] = a[loc];
        a[loc] = temp;
    }

    printArray(a, n);
}

//cocktail random
void cocktailr(int a[], int n)
{
    printArray(a, n);
    cout << endl;

    srand(time(NULL));

        bool swapped = true;
        int start = 0;
        int end = n - 1;

        while (swapped) {
            swapped = false;

            for (int i = start; i < end; ++i) {
                if (a[i] > a[i + 1]) {
                    swap(a[i], a[i + 1]);
                    swapped = true;
                }
            }

            if (!swapped)
                break;

            swapped = false;
            --end;
            for (int i = end - 1; i >= start; --i) {
                if (a[i] > a[i + 1]) {
                    swap(a[i], a[i + 1]);
                    swapped = true;
                }
            }
            ++start;
        }

    printArray(a, n);
}

//merge random
void mergeauxr(int array[], int const left, int const mid, int const right)
{
    auto const subArrayOne = mid - left + 1;
    auto const subArrayTwo = right - mid;

    // Create temp arrays
    auto* leftArray = new int[subArrayOne],
        * rightArray = new int[subArrayTwo];

    // Copy data to temp arrays leftArray[] and rightArray[]
    for (auto i = 0; i < subArrayOne; i++)
        leftArray[i] = array[left + i];
    for (auto j = 0; j < subArrayTwo; j++)
        rightArray[j] = array[mid + 1 + j];

    auto indexOfSubArrayOne = 0, // Initial index of first sub-array
        indexOfSubArrayTwo = 0; // Initial index of second sub-array
    int indexOfMergedArray = left; // Initial index of merged array
    // Merge the temp arrays back into array[left..right]
    while (indexOfSubArrayOne < subArrayOne && indexOfSubArrayTwo < subArrayTwo) {
        if (leftArray[indexOfSubArrayOne] <= rightArray[indexOfSubArrayTwo]) {
            array[indexOfMergedArray] = leftArray[indexOfSubArrayOne];
            indexOfSubArrayOne++;
        }
        else {
            array[indexOfMergedArray] = rightArray[indexOfSubArrayTwo];
            indexOfSubArrayTwo++;
        }
        indexOfMergedArray++;
    }
    // Copy the remaining elements of
    // left[], if there are any
    while (indexOfSubArrayOne < subArrayOne) {
        array[indexOfMergedArray] = leftArray[indexOfSubArrayOne];
        indexOfSubArrayOne++;
        indexOfMergedArray++;
    }
    // Copy the remaining elements of
    // right[], if there are any
    while (indexOfSubArrayTwo < subArrayTwo) {
        array[indexOfMergedArray] = rightArray[indexOfSubArrayTwo];
        indexOfSubArrayTwo++;
        indexOfMergedArray++;
    }
}

void merger(int array[], int const begin, int const end)
{
    if (begin >= end)
```

```
        return; // Returns recursively

    auto mid = begin + (end - begin) / 2;
    merger(array, begin, mid);
    merger(array, mid + 1, end);
    mergeauxr(array, begin, mid, end);
}

//quick random
int partitionr(int arr[], int start, int end)
{

    int pivot = arr[start];

    int count = 0;
    for (int i = start + 1; i <= end; i++) {
        if (arr[i] <= pivot)
            count++;
    }

    // Giving pivot element its correct position
    int pivotIndex = start + count;
    swap(arr[pivotIndex], arr[start]);

    // Sorting left and right parts of the pivot element
    int i = start, j = end;

    while (i < pivotIndex && j > pivotIndex) {

        while (arr[i] <= pivot) {
            i++;
        }

        while (arr[j] > pivot) {
            j--;
        }

        if (i < pivotIndex && j > pivotIndex) {
            swap(arr[i++], arr[j--]);
        }
    }

    return pivotIndex;
}

void quickr(int arr[], int start, int end)
{

    // base case
    if (start >= end)
        return;

    // partitioning the array
    int p = partitionr(arr, start, end);

    // Sorting the left part
    quickr(arr, start, p - 1);

    // Sorting the right part
    quickr(arr, p + 1, end);
}

//heap random
```

```
void heapifyr(int arr[], int n, int i)
{
    int largest = i; // Initialize largest as root
    int l = 2 * i + 1; // left = 2*i + 1
    int r = 2 * i + 2; // right = 2*i + 2

    // If left child is larger than root
    if (l < n && arr[l] > arr[largest])
        largest = l;

    // If right child is larger than largest so far
    if (r < n && arr[r] > arr[largest])
        largest = r;

    // If largest is not root
    if (largest != i) {
        swap(arr[i], arr[largest]);

        // Recursively heapify the affected sub-tree
        heapifyr(arr, n, largest);
    }
}

// main function to do heap sort
void heapr(int arr[], int n)
{
    // Build heap (rearrange array)
    for (int i = n / 2 - 1; i >= 0; i--)
        heapifyr(arr, n, i);

    // One by one extract an element from heap
    for (int i = n - 1; i >= 0; i--) {
        // Move current root to end
        swap(arr[0], arr[i]);

        // call max heapify on the reduced heap
        heapifyr(arr, i, 0);
    }
}

int main()
{
    // -------- Creacion de numeros aleatorios ---------
    int a[2000];
    int n = 2000;

    for (int i = 0; i < n; i++)
    {
        a[i] = rand() % (100 - 1 + 1) + 1;
    }
    //---------- Algoritmos ----------------------------
    unsigned t0, t1;
    t0 = clock();

    //boobler(a,n);
    //insertr(a,n);
    //selectionr(a,n);
    //cocktailr(a,n);

    /*
    printArray(a, n);
    cout << endl;
    merger(a, 0, n - 1);
    printArray(a, n);
    */

    /*
    printArray(a, n);
    cout << endl;
    quickr(a, 0, n - 1);
    printArray(a, n);
    */

    printArray(a, n);
    cout << endl;
    heapr(a, n);
    printArray(a, n);


    t1 = clock();

    double time = (double(t1 - t0) / CLOCKS_PER_SEC);
    cout << endl << "Execution Time (s): " << time << endl;

    return 0;
}
```