Análisis y Algoritmos

Roberto Charreton Kaplun
Universidad de Artes Digitales

Guadalajara, Jalisco

Email: idv17c.rcharreton@uartesdigitales.edu.mx

Profesor: Efraín Padilla

Mayo 27, 2019

1) Merge Sort
El algoritmo merge es un algoritmo de ordenamiento externo estable basado en la técnica divide y vencerás. Es de complejidad O(n log n)

Desarrolle el codigo separando en dos vectores los valores que necesitan ajustarce en la rama de cada lado posteriormente analice el vector con los multiples casos Ascendente, Descente o Random, de estos dos obtuve el mejor y el peor caso en su benchmarking.

Master Method: O(n log n)

Codigo

```
void CManager::benchMark(vector<int> Vector)
{
        start = omp_get_wtime();
        Vector;
        end = omp_get_wtime();
        cout << "Time: " << end - start << " seconds" << endl;

        ficheroSalida.open("InsertBest.txt",ios::app);
        ficheroSalida << end - start << endl;
        ficheroSalida.close();
}

void CManager::benchMarkSerch(int i)
{
        start = omp_get_wtime();
        i;
        end = omp_get_wtime();
        cout << "Time: " << end - start << " seconds" << endl;

        ficheroSalida.open("Serch.txt", ios::app);
        ficheroSalida << end - start << endl;
        ficheroSalida.close();
}
```

2) Quick Sort
Quick Sort es el algoritmo de ordenación más rápido conocido, su tiempo de ejecución promedio es O(n log (n)), siendo en el peor de los casos O(n2), caso altamente improbable.

Desarrolle el codigo partiendo el vector y reescribiendo su informacion, posteriormente hize la funcion recorsiva con el vector con los multiples casos Ascendente, Descente o Random, de estos dos obtuve el mejor y el peor caso en su benchmarking.

Master Method: T(n)=2T(n/2) + (n) T(n) = (n logn)
Codigo

```
int CManager::partition(std::vector<int> &vec, int low, int high)
{
	int pivot = vec[high];	// pivot
	int i = (low - 1);  // Index of smaller element

	for (int j = low; j <= high - 1; j++)
	{

		if (vec[j] <= pivot)
		{
			i++;
			int swap = vec[i];
			vec[i] = vec[j];
			vec[j] = swap;
		}
	}
	int swap = vec[i + 1];
	vec[i + 1] = vec[high];
	vec[high] = swap;
	return (i + 1);
}

vector <int> CManager::quickSort(std::vector<int>& vec, int low, int high)
{
	if (low < high)
	{
		int mid = partition(vec, low, high);

		quickSort(vec, low, mid - 1);
		quickSort(vec, mid + 1, high);
	}
	return vec;
}
```

3) Binary Search
La búsqueda binaria es computada en el peor de los casos en un tiempo logarítmico, realizando O(log n) comparaciones.

Desarrolle el codigo buscando la condicionante del valor asignado, si se encuentra el numero devuelve el indice, de lo contrario devuelve nulo, posteriormente hize la funcion recorsiva con el vector con los multiples casos Ascendente, Descente o Random, de estos dos obtuve el mejor y el peor caso en su benchmarking.

Master Method:O(Log n).
Codigo

```
int CManager::binarySearch(std::vector<int> vec, int l, int r, int value)
{
	if (r >= l) {
		int mid = l + (r - l) / 2;

		if (vec[mid] == value)
			return mid;

		if (vec[mid] > value)
			return binarySearch(vec, l, mid - 1, value);
```

```
                return binarySearch(vec, mid + 1, r, value);
        }


        return -1;
}
```

4) Linear Search
Búsqueda lineal comprueba secuencialmente cada elemento de la lista hasta que encuentra un elemento que coincide con el valor de objetivo. Si el algoritmo llega al fin de la lista sin encontrar el objetivo, la búsqueda termina insatisfactoriamente.

Desarrolle el codigo buscando la condicionante del valor asignado, si se encuentra el numero devuelve el indice, de lo contrario devuelve nulo, posteriormente hize la funcion recorsiva con el vector con los multiples casos Ascendente, Descente o Random, de estos dos obtuve el mejor y el peor caso en su benchmarking.

Master Method:O(n).
Codigo

```
int CManager::linearSearch(std::vector<int> vec, int value)
{

        for (int i = 0; i < vec.size(); i++)
        {
                if (vec[i] == value)
                {
                        return i;
                }
        }
        return -1;
}
```