2020

Diseño de pruebas y análisis de complejidad

SANTIAGO HURTADO SOLIS A003625, CRISTIAN MORALES A00328064, JUAN SEBASTIAN MORALES A00365920

Escenario 1

cliente1 = new Client("Cristian", "Morales", "1101", Tarjet.AHORROS, "123456", 20000, LocalDate.now());

cliente2 = new Client("Santiago", "Hurtado", "4010", Tarjet.AHORROS, "123456", 10000, LocalDate.now());

cliente3 = new Client("Juan", "Morales", "3210", Tarjet.AHORROS, "123456", 30000, LocalDate.now());

Las pruebas por el formato se encuentran en un documento aparte.

Pruebas unitarias corregidas

Explicacion Resolucion Casos de prueba paso a paso por estructura de datos.

Stack

Metodo: push

Escenario y resolucion : Se crea una Pila se agrega el elemento String hola con el metodo Push se verifica con el metodo empty que esta este vacia se utiliza el assert equals esperando false porque la pila no esta vacia.

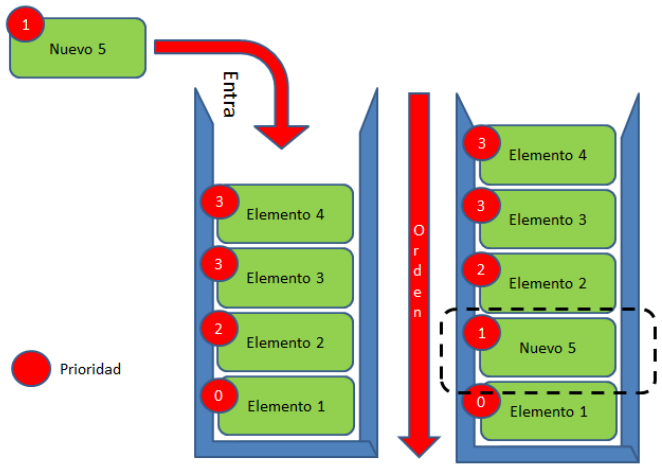
Metodo: pop

Escenario y resolucion: Se crea una Pila se agrega el elemento String hola con el metodo Push, se utiliza el metodo pop para sacar el elmento se verifica que el elmento sea igual al agregado y que la pila este vacia.

Queue

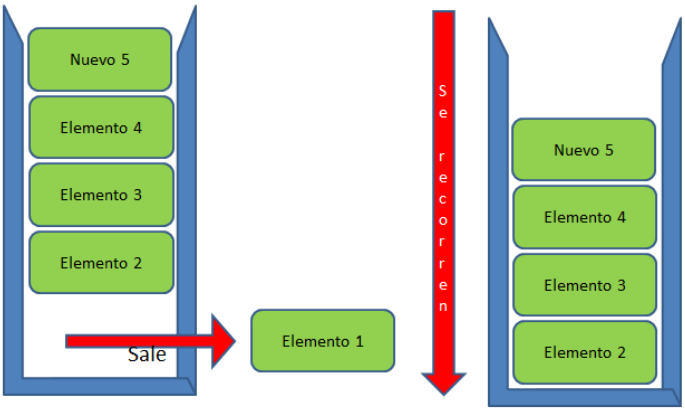
Metodo: queue

Escenario y resolucion: Se crea una cola se agrega un elemento String hola con el metodo enqueue y se verifica con el metodo isEmpty que la cola este vacia utilizando el metodo assert equals que se espera que sea false porque la cola posee un elemento.



Metodo: enqueue

Escenario y resolucion: Se crea una cola se agrega un elemento String hola con el metodo enqueue y se desencola con el metodo enqueue se verifica que el elemento sea igual al ingresao utilizando el metodo equals to el metodo assert equals que se espera que sea true porque son iguales.



Hashtable

Metodo: isEmpty

Escenario y resolucion: se crea una Hash table se utiliza el metodo is empty y se evalua el resultado con el metodo assert equals esperado true porque la tabla esta vacia

Metodo: size

Escenario y resolucion: se crea una hash table se agregan dos tuplas con el metodo add, se evalua el tamaño de la hashtable y se utiliza el metodo assert equals con numero esperado a 2 porque las tuplas que se agregaron fueron dos.

Priority queue

Metodo: enqueue

Escenario y resolucion : Se crea una cola de prioridad , y se encola un elemento String en ella.

Luego se verifica que la cola no este vacia con el metodo isEmpty() ,puesto que este elemento se agrego , posteriormente con el assertsEquals, que se espera que sea falso, porque ya habra un elemento en la priority queue.

Selection

```
private static <T> void exchange(List<T> array, int i, int j) {          //     3
    T aux = array.get(i);                                                //  1
    array.set(i, array.get(j));                                          //  1
    array.set(j, aux);                                                   //  1
}

public  static <T> void selectionSort(List<T> array, Comparator<T> comparator){  //  n = array.lenght

    for (int i = 0; i < array.size(); i++) {                             //  n+1
        int lower = i;                                                   //  n
        for (int j = i+1; j < array.size(); j++) {                       //  n(n+1)/2
            if(comparator.compare(array.get(j),array.get(lower))<0) {    //  n(n+3)/2
                lower = j;                                               //  n(n+3)/2
            }
        }
        if(lower!=i) {                                                   //  n
            exchange(array,lower,i);                                     //  3n
        }
    }
}

                                                                         // SUMA
                                                                         //  (3n^2)/2 + 19n/2
                                                                         //  n(3n+19)/2 <= C*n^2
                                                                         //  O(n^2)
                                                                         //
```

Complejidad Espacial

Selection Sort

	Complejidad espacial	
	variable	cantidad de valores atomicos
Entrada	array	n
salida	resultado	n
auxiliares	lower	1
	i	1
	j	1
resultado		2n+3

Complejidad Temporal Selection

Algoritmo	Linea de codigo	# veces que se repite

Exchange	T aux = array.get(i);
	array.set(i, array.get(j));
	array.set(j, aux);
		3

	for (int i = 0; i < array.size(); i++)	n+1
	int lower = i;	n
Selection	for (int j = i+1; j < array.size(); j++)	n(n+1)/2
	if(comparator.compare(array.get(j),array.get(lower))<0)	n(n+3)/2
	lower = j;	n(n+3)/2
	if(lower!=i)	n
	exchange(array,lower,i);	3n
	TOTAL	O(n^2)

Heapsort

```
27
28⊖   public  static <T> void heapSort(List<T> array, Comparator<T> comparator){
29
30        int n = array.size()-1;
31        while(n>0) {
32            for (int i = (n-1)/2; i >=0; i--) {
33                heapifying(array,i,n,comparator);
34            }
35            exchange(array,0,n);
36            n--;
37        }
38    }
39⊖   private static <T> void heapifying(List<T> array, int i,int size, Comparator<T> comparator) {
40        int left = 2*i +1;
41        int right = 2*i +2;
42        int max=i;
43        if(left<=size) {
44            if(comparator.compare(array.get(left), array.get(i))>0) {
45                max = left;
46            }
47        }
48        if(right<=size) {
49            if(comparator.compare(array.get(right), array.get(max))>0) {
50                max = right;
51            }
52        }
53        if(max!=i) {
54            exchange(array,i,max);
55            heapifying(array,max,size,comparator);
56        }
57    }
58
```

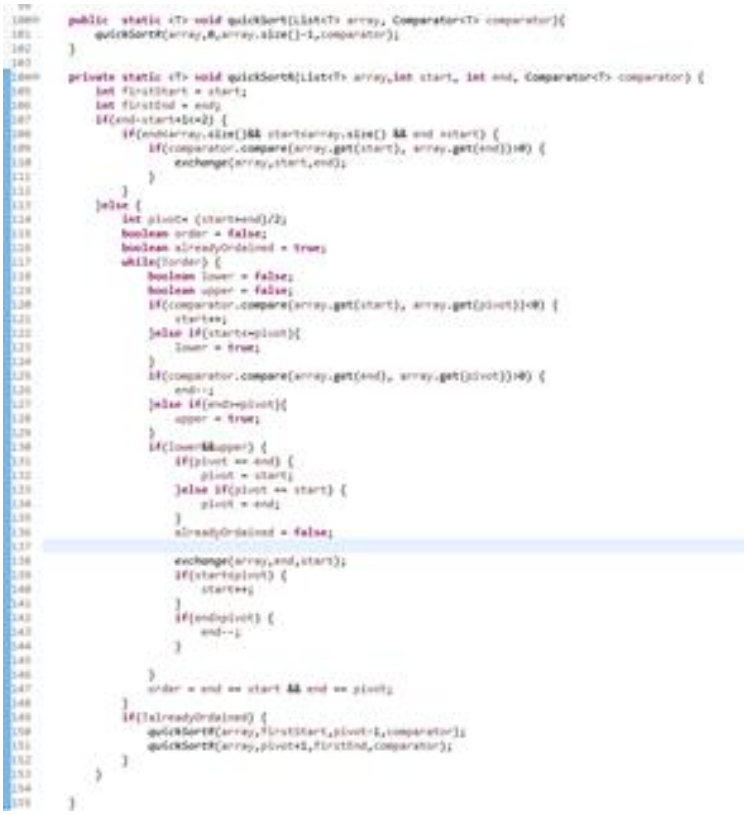
Complejidad temporal

Heap sort	# codigo	# veces que se repite
	int n = array.size()-1;	n
	while(n>0)	n+1
	for (int i = (n-1)/2; i >=0; i--)	n(n+1)
	heapifying(array,i,n,comparator);	lg *n*
	exchange(array,0,n);	3
	n--;	n
Heapifying	int left = 2*i +1;	n
	int right = 2*i +2;	n
	int max=i;	n
	if(left<=size)	n
	if(comparator.compare(array.get(left), array.get(i))>0)	n(n+1)/2
	max = left;	n
	if(right<=size)	n
	if(comparator.compare(array.get(right), array.get(max))>0)	n(n+1)/2
	max = right;	n
	if(max!=i)	Log n
	exchange(array,i,max);	3
	heapifying(array,max,size,comparator);	Log n
Total		O (n log n)

Complejidad Espacial Heap Sort incluyendo variables del heapifying

	Complejidad espacial	
	variable	cantidad de valores atomicos
Entrada	array	n
salida	resultado	n
auxiliares	rigth	1
	size	1
	max	1
	left	1
	i	1
	n	1
resultado	2n+6	

Quiksort



$$t(n) = \begin{cases} 0 & Si \quad n = 1 \\ 2t(n/2) + n & Si \quad n > 1 \end{cases}$$

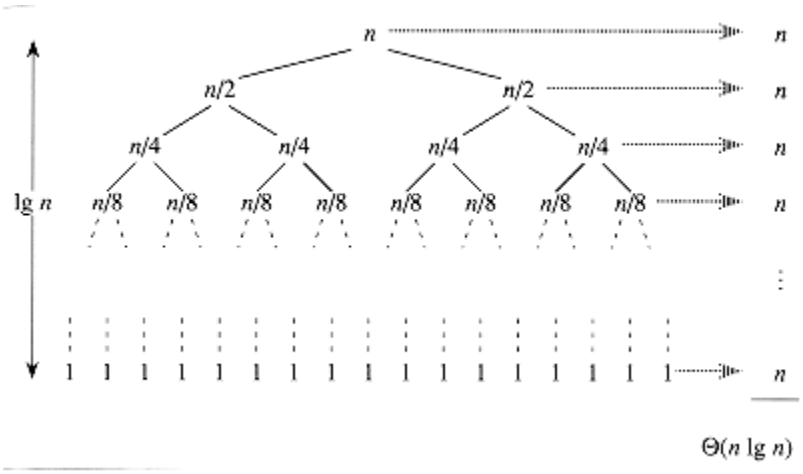
Complejidad espacial Quicksort incluye variables de metodos auxiliares

	Complejidad espacial	
	variable	cantidad de valores atomicos
Entrada	start	1
	end	1
	array	n
salida	resultado	n
auxiliares	start	1
	middle	1
	end	1
	i	1
	j	1
	position	1
	list aux	n
resultado		3n+8

Complejidad temporal del quicksort

Algoritmo	#Codigo	# que se repite cada linea
Quicksort	quickSortR(array,0,array.size()-1,comparator);	1
QuicksortR	int firstStart = start;	1
	int firstEnd = end;	1
	if(end-start+1<=2)	n
	if(end<array.size()&& start<array.size() && end >start)	n
	if(comparator.compare(array.get(start), array.get(end))>0)	n
	exchange(array,start,end);	3
	else	
	int pivot= (start+end)/2;	1
	boolean order = false;	n
	boolean alreadyOrdained = true;	1
	while(!order)	Log n+1
	boolean lower = false;	1
	boolean upper = false;	1
	if(comparator.compare(array.get(start), array.get(pivot))<0)	n
	start++;	n
	else if(start<=pivot)	n
	lower = true;	1
	if(comparator.compare(array.get(end), array.get(pivot))>0)	n
	end--;	n
	else if(end>=pivot)	n
	upper = true;	1

	if(lower&&upper)	n
	if(pivot == end)	n
	pivot = start;	1
	else if(pivot == start)	n
	pivot = end;	1
	alreadyOrdained = false;	1
	exchange(array,end,start);	3
	if(start<pivot)	n
	start++;	n
	if(end>pivot)	n
	end--;	n
	order = end == start && end == pivot;	n
	if(!alreadyOrdained)	n
	quickSortR(array,firstStart,pivot -1,comparator);	Log n
	quickSortR(array,pivot+1,firstEnd,comparator);	Log n
TOTAL		O(n log n)



```java

    public  static <T> void mergeSort(List<T> array, Comparator<T> comparator){
        mergeSortR(array,0,array.size()-1,comparator);
    }
    public static <T> void mergeSortR(List<T> array, int start,int end,Comparator<T> comparator) {
        if(end-start+1>=2) {
            mergeSortR(array,start,(start+end)/2,comparator);
            mergeSortR(array,((start+end)/2)+1,end,comparator);
        }
        toCombine(array,start,(start+end)/2,end,comparator);
    }
    private static <T> void toCombine(List<T>array, int start, int middle, int end,Comparator<T> comparator) {
        int i = start;
        int j = middle +1;
        List<T> listAux = new ArrayList<T>();
        if((end-start+1)>1) {
            while(i<=middle && j<=end) {

                if(comparator.compare(array.get(i), array.get(j))<0) {
                    listAux.add(array.get(i));
                    i++;
                }else {
                    listAux.add(array.get(j));
                    j++;
                }
            }
            for (int k = i; k <= middle; k++) {
                listAux.add(array.get(k));
            }
            for (int k = j; k <= end; k++) {
                listAux.add(array.get(k));
            }
        }
        int position = start;
        for (int k = 0; k < listAux.size(); k++) {

            array.set(position,listAux.get(k));
            position++;
        }

    }
```

Complejidad espacial Merge Sort

	Complejidad espacial	
	variable	cantidad de valores atomicos
Entrada	start	1
	end	1
	array	n
salida	resultado	n
	start	1
	middle	1
	end	1
auxiliares	i	1
	j	1
resultado		2n+8

Complejidad temporal merge sort

Algoritmo	Linea de codigo	# veces que se repite
Mergesort	mergeSortR(array,0,array.size()-1,comparator);	1
MergeSortR	if(end-start+1>=2)	n
	mergeSortR(array,start,(start+end)/2,comparator);	n
	mergeSortR(array,((start+end)/2)+1,end,comparator);	n
	toCombine(array,start,(start+end)/2,end,comparator);	2x^2+10+9
To combine	int i = start;	1
	int j = middle +1;	1
	List<T> listAux = new ArrayList<T>();	1
	if((end-start+1)>1)	n
	while(i<=middle && j<=end)	n+1
	if(comparator.compare(array.get(i), array.get(j))<0)	n(n+1)
	listAux.add(array.get(i));	n
	i++;	n
	else	
	listAux.add(array.get(j))	n
	j++;	n
	for (int k = i; k <= middle; k++)	n(n+1)
	listAux.add(array.get(k))	1
	for (int k = j; k <= end; k++)	n
	listAux.add(array.get(k))	1
	int position = start;	1
	for (int k = 0; k < listAux.size(); k++)	n+1
	array.set(position,listAux.get(k));	1
	position++;	n
TOTAL		2x^2+10+9= N (n log n)