Gebze Technical University
Computer Engineering

CSE 222 - 2018 Spring

HOMEWORK 4 Parts 1-4

ÖMER ÇEVİK
161044004

Course Assistant: Ayşe Şerbetçi TURAN

PART 1

a-)

```
public List iterativeSearch(List L)
{
	// It finds current sublist first index and size, next sublist first index and size.
	// Then compares that sizes which is maximum.
	// It keeps doing finding the maximum sized sublist and using index accessing it.
	// Adding the result list and returns.

	ListHead Lhead = L.head;
	ListHead LH = L.head;
	ListHead Saver = new ListHead();
	List Result = new List();

	int CurrentIndex = 0, NextIndex = 0, MaxIndex = 0;
	int CurrentDataSize = 1, NextDataSize = 1, MaxSize = 1;
	int Size = 1;

	while(Lhead != null)
	{
		if(Lhead.next != null)
		{
			if(Lhead.next.data >= Lhead.data)
			{
				++CurrentDataSize;
			}
			else if(Lhead.next.data < Lhead.data)
			{
				NextIndex = Size;
				ListHead NextHead = Lhead.next;
				while(NextHead != null)
				{
					if(NextHead.next != null)
					{
						if(NextHead.next.data >= NextHead.data)
						{
							++NextDataSize;
						}
						else
						{
							if(CurrentDataSize <= NextDataSize)
							{
								MaxIndex = NextIndex;
								Size = CurrentIndex;
								MaxSize = NextDataSize;
								NextDataSize = 1;
							}
```

```
                                                    else
                                                    {
                                                            MaxIndex = CurrentIndex;
                                                            Size = NextIndex;
                                                            MaxSize = CurrentDataSize;
                                                            CurrentDataSize = 1;
                                                    }
                                                    Lhead = NextHead;
                                                    break;
                                            }
                                    }
                                    NextHead = NextHead.next;
                            }
                    }
                    }
                    ++Size;
                    Lhead = Lhead.next;
            }
            while(MaxIndex > 0)
            {
                    LH = LH.next;
                    --MaxIndex;
            }
            Result.head = Saver;
            Saver = LH;
            while(MaxSize > 0)
            {
                    Saver = Saver.next;
                    --MaxSize;
            }
            Saver.next = null;
            return Result;
    }
```

Time Complexity:

T(n) = 15+n+1+max(1+1,1+2+1+max(3,1+max(4,4))) + MaxIndex + MaxSize + CurrentSubListSize + NextSubListSize

= n + 15 + 1 + 9 + MaxIndex + MaxSize + CurrentSubListSize + NextSubListSize = n

= O(n)

b-)

```
public List recursivelySearch(List L)
{
        if (List.head != null)
        {
                List CurrentList = new List();
                CurrentList.head = L.head;
                ListHead CurrentSubListHead = CurrentList.head;
                int CurrentSize = 1;
                while (L.head !=null)
                {
                        if(L.head.next != null)
                        {
                                if( L.head < L.head.next)
                                {
                                        L.head = L.head.next ;
                                        CurrentSubListHead.next = L.head;
                                        CurrentSubListHead = CurrentSubListHead.next;
                                        ++CurrentSize;
                                }
                        }
                }
                CurrentSubListHead.next = null;

                List Result = recursivelySearch (L);

                ListHead NextSubListHead = Result.head;
                int NextSize = 1;
                while ( NextSubListHead != null)
                {
                        NextSubListHead = NextSubListHead.next;
                        ++NextSize;
                }
                NextSize > CurrentSize ? Result : CurrentList ;
        }
        else
                return L;
}
```

<u>Time Complexity:</u>

$T(n) = 1$

$T(n) = 1+4+\max(0,\max(0,4))+1+T(n/1)+3+NextSubListSize + \max(1,1)$

$= T(n/1) + 13 + NextSubListSize = T(n/1)$

Master Theorem used: $a = 1, b = 1, d = 0;\ a = b^d;\quad 1 = 1\text{^}0$ then, $T(n) = \Theta(n^d \log n) = \Theta(\log n)$

Induction used: $n > 0, c > 0; n_0 = 1, c = 1$; then $T(n) = 1$ is true.

for $n = k, T(n) = c(T(k) + 13 + NextSubListSize)$, we assume that it is true

for $n = k+1$, then $T(n) = c(T(k+1) + 13 + NextSubListSize)$

$T(n) = T(k+1) + 13 + NextSubListSize$

$n = k-1$ then $T(k-1) = T(k) + 13 + NextSubListSize$, then it is true.

PART 2

```
private static final int N = 8;
int[] arr = {1, 2, 3, 5, 7, 9, 15, 22};
int front = 0, rear = N-1, x = 12;

while(front < rear)
{
        if (arr[front] + arr[rear] == x)
        {
                System.out.printf("%d = %d + %d\n",x, arr[front],arr[rear] );
                break;
        }
        else if ( arr[front] + arr[rear] > x)
        {
                --rear;
        }
        else
                ++front;
}
```

Analyze Algorithm

We know the array size N. This array saves first index and last index as front and rear.
In while loop we check if array's front and rear indexed datas sum is equal to x. If it is then we found our values in array. Otherwise if array's front and array's rear indexed values sum is greater than x then we know that in ascending sorted list's rear indexed value is greater. So rear index must be decreased. Otherwise if array's front and array's rear indexed values sum is smaller then we must increase front index. Then it goes in $\Theta(n)$ time complexity. Because it walks in N sized array only once.

PART 3

```
for (i=2*n; i>=1; i=i-1)
        for (j=1; j<=i; j=j+1)
                for (k=1; k<=j; k=k*3)
                        print("hello")
```

Time Complexity: $T(n) = (\Sigma_{i=1}^{2n}(\Sigma_{j=1}^{i} j/3 + 1)) = j*(j+1)/6 + j = 1/6\Sigma_{i=1}^{2n}(j^2+7j)$

$= (2*n(2*n+1)/2)(\Sigma_{j=1}^{2*n}\log_3(j))$

$= n*(2*n+1)(\Sigma_{j=1}^{2n}\log_3(j))$

$= (2n^2+n)(\Sigma_{j=1}^{2n}\log_3(j))$

$= O(n^2(\Sigma_{j=1}^{2n}\log(j)))$

PART 4

```
float aFunc(myArray,n){
    if (n==1){
        return myArray[0];
    }

    for (i=0; i <= (n/2)-1; i++){
        for (j=0; j <= (n/2)-1; j++){
                myArray1[i]  =  myArray[i];
                myArray2[i]  =  myArray[i+j];
                myArray3[i]  =  myArray[n/2+j];
                myArray4[i]  =  myArray[j];
        }
    }
    x1 = aFunc(myArray1,n/2);
    x2 = aFunc(myArray2,n/2);
    x3 = aFunc(myArray3,n/2);
    x4 = aFunc(myArray4,n/2);

    return x1*x2*x3*x4;
}
```

Using Master Theorem:

$T(n) = 1$

$T(n) = 1 + \max(1, (n/2)*(n/2)*4 + T(n/2) + T(n/2) + T(n/2) + T(n/2) + 1)$

$= 1 + n^2 + 4T(n/2) + 1 = 4T(n/2) + n^2 + 2$

$T(n) = aT(n/b) + n^d$, then $a = 4, b = 2, d = 2$.

$a = b^d$, $4 = 2^2$ then $T(n) = \Theta(n^d \log n) = \Theta(n^2 \log n)$