

**ECU178 Computer Science:
210CT - Programming, Algorithms and Data
Structures Portfolio**

Due on Monday, December 15th, 2014

Dr James Shuttleworth

Robert Rigler : 4939377

Contents

Item 1: Week 3 - Linear Search and Duplicate Finder	3
1. Pseudocode for linear search	3
2. Pseudocode for finding duplicates in a list	3
Item 2: Week 4 - Time complexities and Big-O notation	4
1. Describe the runtime bounds of the linear search algorithm	4
2. Describe the runtime bounds of the duplicate finder algorithm	4
Additional work: Critical values of relative runtimes	5
Item 3: Week 6 - Harmonic Series	6
1. Harmonic Series (Pseudocode)	6
2. Harmonic Series (JAVA Implementation)	7
Item 4: Week 7 - Heapworksheet or RPN Calculator	9
Item 5: Week 8 - Linked List	10
1. Linked List Implementation	10

Item 1: Week 3 - Linear Search and Duplicate Finder

1. Pseudocode for linear search

This Simple Algorithm demonstrates how to perform a linear search.

Input: This algorithm takes a populated array A and a value to search for v , as parameters.

Output: The Algorithm is a boolean type and returns either True or False respective of whether the v was found in the list or not.

Algorithm 1 LinearSearch

```
1: procedure BOOL LINEARSEARCH( $v$ ,  $A[]$ )
2:   for each element  $i$  in  $A$  do
3:     if  $A[i] = v$  then
4:       return true
5:     end if
6:   end for
7: return false
8: end procedure
```

2. Pseudocode for finding duplicates in a list

This algorithm demonstrates how to examine if a list has duplicate entries using a linear search.

Input: This algorithm takes a populated array A as a parameter.

Output: This Algorithm is a boolean type and returns true or false respective of whether a duplicate value is found or not.

Algorithm 2 Examining for duplicates

```
1: procedure BOOL EXFORDUPES( $A[]$ )
2:   for each element  $i$  in  $A[]$  do
3:     for each element  $j$  in  $A[]$  do
4:       if  $A[i] = A[j]$  then
5:         return true
6:       end if
7:     end for
8:   end for
9: end procedure
```

Item 2: Week 4 - Time complexities and Big-O notation

1. Describe the runtime bounds of the linear search algorithm

Algorithm 3 LinearSearch

```

1: procedure BOOL LINEARSEARCH(item, list[ ])
2:
3:   for each element i in list do           (n)
4:     if list[i] = item then t           (n)
5:       return true                         (n)
6:     end if
7:   end for
8: return false                             (1)
9: end procedure

```

Collecting the line-by-line runtime data from the algorithms gives: $n + n + n + 1$ which is equivalent to: $3n + 1$.

Therefore the time complexity of the algorithm is $O(n)$.

2. Describe the runtime bounds of the duplicate finder algorithm

Algorithm 4 Examining for duplicates

```

1: procedure BOOL EXFORDUPES(list[ ])
2:   for each element i in list[ ] do           (n)
3:     for each element j in list[ ] do           (n*n)
4:       if list[i] = list[j] then           (n*n)
5:         return true                         (n*n)
6:       end if
7:     end for
8:   end for
9: return false                             (1)
10: end procedure

```

Collecting the line-by-line runtime data from the algorithms gives: $n + (n * n) + (n * n) + (n * n) + 1$ which is equivalent to: $3n^2 + n + 1$.

Therefore the time complexity of the algorithm is $O(n^2)$.

Additional work: Critical values of relative runtimes

Write a function that determines the critical value at which the relative runtime of two linear algorithms swap.

For this algorithm, I am assuming that $k_1 > k_2$ (Expression 1 > Expression 2, when $n = 0$). The Algorithm is very simple; While the value of Expression 1 is greater than Expression 2, increase the value of n .

When the Runtime of the algorithms swap, the while-loop exit condition is fulfilled and the current value of n is returned.

Algorithm 5 Relative runtime comparison algorithm

```
1: procedure CRITVAL( $m_1, k_1, m_2, k_2$ )
2:   while ( $m_1 * n + k_1$ ) > ( $m_2 * n + k_2$ ) do
3:      $n++$ 
4:   end while
5: return  $n$ 
6:
7: end procedure
```

Item 3: Week 6 - Harmonic Series

1. Harmonic Series (Pseudocode)

Use pseudocode to specify a recursive algorithm to compute the n th value of the harmonic series, for some integer n .

The Harmonic series is as follows: $1 + 1/2 + 1/3 + 1/4 + 1/5 + \dots 1/n$

Input: This algorithm takes two parameters t and n which are the total sum of the algorithm and the number of repetitions, respectively.

Output: This algorithm outputs the value t which is the total sum of the harmonic series.

This procedure uses a while-loop to control the number recursive iterations.

While the number of iterations left is above 0, add the next value to t , decrease the number of iterations by 1 and recursively call the procedure with the new values of t and n .

When the number of iterations left is no longer above 0, the final value of t is returned and the procedure ends.

Algorithm 6 Computing n th value of harmonic series

```
1: procedure HARM(float  $t$ , float  $n$ )
2:   while  $n > 0$  do
3:      $t \leftarrow t + (1/n)$ 
4:      $n \leftarrow n - 1$ 
5:     HARM( $t, n$ )
6:   end while
7: return  $t$ 
8: end procedure
```

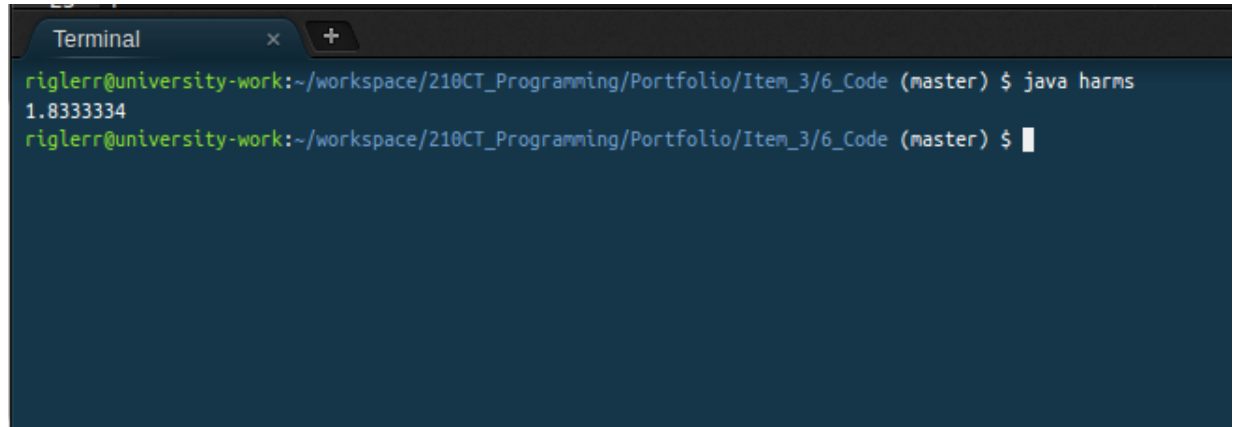
2. Harmonic Series (JAVA Implementation)

The Harmonic Series computation algorithm implemented in Java

Listing 1: harms java class file

```
1 public class harms{
2
3     public static void main(String[] args){
4         /**/
5         System.out.println(f(0,3));
6     }
7
8     public static float f(float t, float n){
9         /*
10         t always has a value of 0 on the initial method call.
11         n is the nth term, which decreases by 1 each recursive call.
12
13         When n = 0, stop recursive calling and return the value t.
14         */
15         while (n>0)
16         {
17             t+= (1/n);
18             f(t,--n);
19         }
20         return t;
21     }
22
23
24 }
```

Evidence of the Harmonic Series computation java implementation.
The nth value passed to the method was 3.



```
Terminal
riglerr@university-work:~/workspace/210CT_Programming/Portfolio/Item_3/6_Code (master) $ java harms
1.8333334
riglerr@university-work:~/workspace/210CT_Programming/Portfolio/Item_3/6_Code (master) $
```


Item 4: Week 7 - Heapworksheet or RPN Calculator

Item 5: Week 8 - Linked List

1. Linked List Implementation

Below are the Linked List and Node classes that I have created as well as a Test class.

This is a fully functional doubly Linked List capable of Inserting, Prepending, Deleting and Displaying items in the List. Please see the Evidence Screenshot below:

Listing 2: List Class Implementation: Java

```
1 package com.company;
2 public class List {
3     //Nodes to hold the head and tail of the list respectively
4     Node head;
5     Node tail;
6     //-----
7     //-----List Constructor to init values to null-----
8     //-----
9     public List(){
10         head = null;
11         tail = null;
12     }
13     //-----
14     //----Insert an Element After the specied element (N)-----
15     //-----
16     public void Insert(Node n, Node x){
17         //If previous node exists
18         if(n != null){
19             //SWAP NODE POINTERS
20             x.next = n.next;
21             n.next = x;
22             x.prev = n;
23
24             // If new node isn't end of list
25             if(x.next != null)
26                 x.next.prev = x;
27         }
28         //Prepend to existing list
29         if (head != null & n == null) {
30             n = head;
31             n.prev = x;
32             x.next = n;
33             x.prev = null;
34             head = x;
35         }
```

```
36
37
38
39     // If new list (First item to be placed)
40     else if (n == null & head == null)
41     {
42         //make new node equal to both head and tail
43         head = tail =x;
44         x.prev =x.next = null; // make head pointers null
45     } //If all else exp
46     else if (tail == n)
47         tail =x;
48 }
49 //-----
50 //-----DELETE A NODE FROM THE LIST-----
51 //-----
52 public void Delete(Node n){
53     //create temporary node
54     Node i = head;
55
56     while (i != null){ // Loop until null (tail.next)
57
58         if (i.value == n.value){
59
60             if(i == head){ //If node to remove is head node
61                 head = i.next;
62                 i.prev = null;
63                 break;
64             }
65
66             //if node to remove is tail node
67             else if (i == tail){
68                 tail = i.prev;
69                 i.prev.next = null;
70             }
71             else { // If node to remove is not Head/Tail
72                 i.prev.next = i.next;
73                 i.next.prev = i.prev;
74                 break; }
75         }
76         i=i.next; // Increment
77     }
78 }
```

```
79
80
81
82 //-----
83 //-----DISPLAY ALL NODES IN LIST-----
84 //-----
85 public void display(){
86     Node i = head;
87
88     while(i != null) {
89         System.out.print(i.value + ", ");
90         i = i.next;
91     }
92 }
93
94
95 }
```

Listing 3: Node Class Implementation: Java

```
1 package com.company;
2
3 /**
4  * Created by rob on 20/12/14.
5  */
6
7 public class Node {
8     int value;
9     Node prev;
10    Node next;
11
12    public Node(int val){
13
14        //Node Constructor
15        value = val;
16        next =null;
17        prev =null;
18
19    }
20
21
22 }
```

Listing 4: Test class

```
1 package com.company;
2
3 public class Main {
4
5     public static void main(String[] args) {
6         List obj1 = new List();
7
8
9         obj1.Insert(null, new Node(42));
10        obj1.Insert(obj1.head, new Node(32));
11        obj1.Insert(obj1.head.next, new Node(102));
12
13        //-----
14        //-----Test Inserting Tail-----
15        //-----
16        System.out.println("Insert 3 items:");
17        obj1.display();
18        System.out.println();
19
20        // -----
21        //-----Test Insert Between-----
22        //-----
23        //Insert 12 between the head node and the head.next
24        System.out.println("Insert 12 after head node: ");
25        obj1.Insert(obj1.head, new Node(12));
26        obj1.display();
27        System.out.println();
28
29        //-----
30        //-----Test Prepend-----
31        //-----
32        //Insert 13 at start of pre-existing list
33        System.out.println("Prepend 13: ");
34        obj1.Insert(null, new Node(13));
35        obj1.display();
36        System.out.println();
37
38
39
40
41
42
```

```
43      //-----  
44      //-----Test Delete Head-----  
45      //-----  
46      //Delete the head node '13' from the list  
47      System.out.println("Delete Head: ");  
48      obj1.Delete(new Node(13));  
49      obj1.display();  
50      System.out.println();  
51  
52      //-----  
53      //-----Test Delete Norm-----  
54      //-----  
55      //Delete a middle node (32) from list an display  
56      System.out.println("Delete a Middle Node (32)");  
57      obj1.Delete(new Node(32));  
58      obj1.display();  
59      System.out.println();  
60  
61      //-----  
62      //-----Test Delete Tail-----  
63      //-----  
64      //Delete the tail node '42' from list an display  
65      System.out.println("Delete tail: ");  
66      obj1.Delete(new Node(102));  
67      obj1.display();  
68      System.out.println();  
69  
70  
71      }}
```

Evidence of working linked list using the Test class shown above:

```
/usr/lib/jvm/java-8-oracle/bin/java ...  
Insert 3 items:  
42, 32, 102,  
Insert 12 after head node:  
42, 12, 32, 102,  
Prepend 13:  
13, 42, 12, 32, 102,  
Delete Head:  
42, 12, 32, 102,  
Delete a Middle Node (32)  
42, 12, 102,  
Delete tail:  
42, 12,
```