CSCI 2110 Data Structures and Algorithms

Module 4: Ordered Lists

DALHOUSIE UNIVERSITY

Learning objectives

- ***Define the ordered list data structure.***

- ***Develop binary search algorithm on ordered lists.***

- ***Build an OrderedList class.***

- ***Understand merging operations on ordered lists.***

What is an ordered list?

- *It is a linear collection of items, in which the items are arranged in either ascending or descending order of **keys**.*
- *The **key** is one part of the item. It serves as the basis of ordering. The key may vary from application to application.*
- *This means that an ordered list is **sorted and maintained sorted** - even when items are added or deleted.*
- *Repetition of items is normally **not allowed**.*
- *Example of an ordered list: List of student entries in a particular course (student name, ID number, major, marks). Here the key could be the student name or the ID number.*

In the first list, the key is the name;

In the second list, the key is the ID no.

Sorted

Name	ID	Major	Mark
Andy	9856	BCS	90.5
Boris	7859	BInf	87.5
Dominic	3664	BA	96.6
Earl	5654	BCS	77.6
Tasha	8776	BSc.	93.4

item

Sorted

Name	ID	Major	Mark
Dominic	3664	BA	96.6
Earl	5654	BCS	77.6
Boris	7859	BInf	87.5
Tasha	8776	BSc.	93.4
Andy	9856	BCS	90.5

What is the advantage of having a list that is always ordered?

- The big advantage of an ordered list is that it enables fast search for an item.
- This is because ordered lists can be searched using the Binary Search algorithm.
- Binary search of **n items** takes only **$O(\log_2 n)$** in the worst case.
- We will see, however, that this advantage comes with a cost – for inserting and deleting items.

Binary Search Algorithm

- Binary search is a powerful algorithm that can be used to search for a key in a sorted list with non-repeated items.

- *The idea in binary search is to divide the list in half, and check if the item is in the left half or the right half.*

- This procedure is repeated on smaller and smaller portions of the list.

Binary Search Example

Search for key = 59 in the following array:

4	**5**	**9**	**12**	**18**	**21**	**24**	**27**	**28**	**32**	**45**	**59**	**60**
0	1	2	3	4	5	6	7	8	9	10	11	12

Check the element in the middle of the array.
The middle element is a[6] = 24.
The key 59 is > 24
So if the key is present, it should be in the right half.

Binary Search Algorithm

4	5	9	12	18	21	24	27	28	32	45	59	60
0	1	2	3	4	5	6	7	8	9	10	11	12

Check the element in the middle of this half.

The element is a[9] = 32

The key 59 is > 32

So if the key is present, it should be in the right half of this half.

Binary Search Algorithm

4	5	9	12	18	21	24	27	28	32	45	59	60
0	1	2	3	4	5	6	7	8	9	10	11	12

Check the element in the middle of this half.
The element is a[11] = 59.
Key found!

BINARY SEARCH ALGORITHM IN MORE DETAIL

Let's understand binary search algorithm in more detail. Assume that the ordered list consists of Strings (names) stored in an array. The principle will be the same for binary search on any other data structure and for any other type of object.

Search for "Dan" → target key

Amar	Boris	Charlie	Dan	Fujian	Inder	Liz	Sam	Travis	Wendy
0	1	2	3	4	5	6	7	8	9

↑ lo ↑ hi

lo = 0, hi = 9. mid = (lo + hi)/2 = (0+9)/2 = 4

Integer division

Check the item at index 4. Dan < Fujian (alphabetically)

∴ Check the left half.

Amar	Boris	Charlie	Dan	Fujian	Inder	Liz	Sam	Travis	Wendy
0	1	2	3	4	5	6	7	8	9

↑ lo ↑ mid ↑ hi

lo = 0, hi = 3, mid = (lo+hi)/2 = (0+3)/2 = 1

Check the item at index 1. Dan > Boris.

∴ Check the right half.

Amar	Boris	Charlie	Dan	Fujian	Inder	Liz	Sam	Travis	Wendy
0	1	2	3	4	5	6	7	8	9

↑ lo ↑ hi

lo = mid + 1 = 1 + 1 = 2, hi = 3

New mid. mid = $\frac{2+3}{2}$ = 2 Dan > Charlie.

∴ Search the right half.

Amar	Boris	Charlie	Dan	Fujian	Inder	Liz	Sam	Travis	Wendy
0	1	2	3	4	5	6	7	8	9

lo = mid + 1 = 2+1 = 3, hi = 3

∴ mid = $\frac{3+3}{2}$ = 3 Dan found!

Another example: Search for "Trevor"

Amar	Boris	Charlie	Dan	Fujian	Inder	Liz	Sam	Travis	Wendy
0	1	2	3	4	5	6	7	8	9

lo	hi	mid	Found?
0	9	$(0+9)/2 = 4$	No. Trevor > Fujian. Go right
5	9	$(5+9)/2 = 7$	No. Trevor > Sam. Go right
7+1 = 8	9	$(8+9)/2 = 8$	No. Trevor > Travis. Go right
8+1 = 9	9	$(9+9)/2 = 9$	No. Trevor < Wendy. Go left
9	9-1 = 8		

Stop. lo > hi ∴ Trevor not found.

Pseudocode

```
Algorithm Binary Search (A, n, t)
Input: array A of length n, target t

lo <-- 0
hi <-- n-1
mid <-- (lo+hi)/2
while (lo <= hi)
{
      if (t == A[mid])
           key found; break out of loop

      else if (t < A[mid])
           hi <-- mid-1                 } ← Go left

      else if (t > A[mid])
           lo <-- mid + 1               } ← Go right

      mid <-- (lo+hi)/2                 } ← find the new mid
}

if (lo > hi)
           key not found
else
           key found at mid
```

Complexity Analysis of Binary Search

Size of the list n	Maximum number of searches
n= 16 2^4	5 searches: 1st search: array of size 16; 2nd search: " 8; 3rd search: " 4; 4th search: " 2; 5th search: " 1
n=32 2^5	6 searches $\{ 32 \to 16 \to 8 \to 4 \to 2 \to 1$
n=64 2^6	7 searches $\{ 64 \to \ldots \ldots 1$
n=2^k	K+1 searches

Complexity: n is a power of two

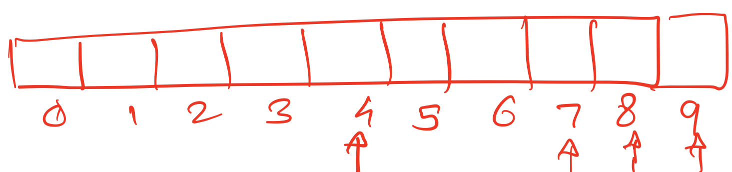
$$\# \text{ searches} = K+1 \quad \text{where } n = 2^K$$

$$= \log_2 n + 1 \quad (\text{that is, } K = \log_2 n)$$

$$\longrightarrow \boxed{O(\log_2 n)}$$

Complexity: n is not a power of two

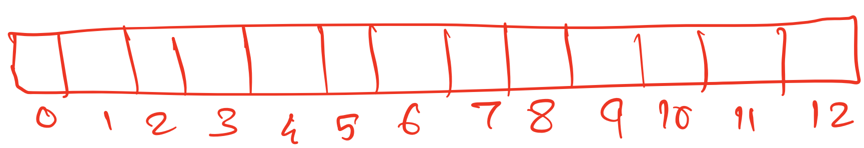
$n = 10$



\# searches = 4

$10 \to 5 \to 2 \to 1$

$n = 13$



\# searches = 4

$13 \to 6 \to 3 \to 1$

$\# \text{ searches} = \lceil \log_2 n \rceil$ e.g. $\lceil \log_2 13 \rceil = \lceil 3.x \rceil \to 4$

$\boxed{O(\log_2 n)}$

Implementation of the OrderedList class

- Our internal data structure to implement the OrderedList will be an ArrayList and not a LinkedList.
- Why? Binary search on a linked list is expensive.
- If we use the ArrayList, search is fast but it comes with a price.
  - Each time we insert an item (in the correct position) or remove an item, entries will have to be shifted.
  - This will cost O(n) for each insert or remove.
  - We will accept this cost because search is the important operation on such lists.

- *We need to write a generic class.*
- *Recall that the entire binary search operation is based on comparisons.*
- *This means we need a generic compareTo method.*
- *Java has a generic interface called Comparable<T> for comparing objects of any type.*
- *It has a method called compareTo with three possible return values:*
  - *0 → equal objects*
  - *Positive integer → "this" object is greater than the parameter*
  - *Negative integer → "this" object is less than the parameter.*

s1.compareTo(s2)

s1 = "Srini" s2 = "Srini"
→ 0

- *Several Java classes like the String class implement this method.*

s1 = "Srini" s2 = "steve"
→ -ve integer

- *If we need compareTo to work for any kind of object, we must define our class as follows:*

public class OrderedList<T extends Comparable<T>> {….}

s1 = "Steve" s2 = "Srini"
→ +ve integer

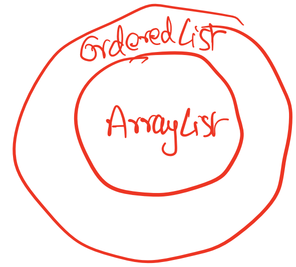
IMPLEMENTATION OF ORDERED LIST CLASS

Constructors

`OrderedList()`	Constructs an empty ordered list

Methods

Name	What it does	Header	Price tag (complexity)
size	returns size of the list	int size()	$O(1)$
isEmpty	returns true if list is empty	boolean isEmpty()	$O(1)$
clear	clears the list	void clear()	$O(1)$
get	gets the entry at the specified position	T get(int pos)	$O(1)$
first	gets the first entry	T first ()	$O(1)$
next	gets the next entry	T next()	$O(1)$
enumerate	scans the list and prints it	void enumerate()	$O(n)$
binarySearch	searches for a given item. returns position (index) if found if not found returns a negative number	int binarySearch(T item)	$O(\log_2 n)$
add	add a specified item at a given position	void add(int pos, T item)	$O(n)$
insert	insert a specified item at the right position	void insert(T item)	$O(n)$
remove	remove a specified item	void remove(T item)	$O(n)$
remove	remove item from a specified position	void remove(int pos)	$O(n)$




```
import java.util.ArrayList;
public class OrderedList<T extends Comparable<T>>
{
      //instance variables
      private ArrayList<T> elements;
      private int cursor;

      //create an empty OrderedList
      public OrderedList()
      {
            elements = new ArrayList<T>();
            cursor=-1;
      }
```

Constructor

```
      //create an empty OrderedList with a given capacity
      //another useful constructor
      public OrderedList(int cap)
      {
            elements = new ArrayList<T>(cap);
            cursor=-1;
      }
```

Overloaded constructor

```
      //returns size of the list
      public int size()
      {
            return elements.size();
      }

      //checks if the list is empty
      public boolean isEmpty()
      {
            return elements.isEmpty();
      }

      //clears the list
      public void clear()
      {
            elements.clear();
      }

      //get the item at a given index
      public T get(int pos)
      {
            if (pos<0||pos>=elements.size())
            {
                  System.out.println("Index out of bounds");
                  return null;
            }
            return elements.get(pos);
      }
```

ArrayList get method

will be used together to step through the orderedlist

```
//Methods first and next are useful to scan the list
//first gets the first item
//next gets the next item (wherever the cursor is)
public T first()
{
      if (elements.size()==0)
            return null;
      cursor=0;
      return elements.get(cursor);
}
public T next()
{
      if (cursor<0||cursor>=(elements.size()-1))
            return null;
      cursor++;
      return elements.get(cursor);
}


//print the list
public void enumerate()
{
      System.out.println(elements);
}



//add an item at a given position
public void add(int pos, T item)
{
      elements.add(pos, item);

}
```

```
//binary search
public int binarySearch(T item)
{
    if (elements.size() == 0) return -1;
    int lo = 0, hi = elements.size() - 1, mid = 0;
    while (lo <= hi)
    {
        mid = (lo + hi) / 2;
        int c = item.compareTo(elements.get(mid));
        if (c == 0) return mid;
        if (c < 0) hi = mid - 1;
        if (c > 0) lo = mid + 1;
    }
    if (item.compareTo(elements.get(mid)) < 0)
        return (-(mid + 1));
    else
        return (-(mid + 2));
}
```

WHY?

EXAMPLE:

B	C	E	G
0	1	2	3

B	C	D	E	G
0	1	2	3	4

Take -3 → convert to +3
subtract 1

binarySearch("E") → 2
binarySearch("D") → -3
binarySearch("F") → -4
binarySearch("H") → -5
binarySearch("A") → -1

```
//insert an item at the correct position
public void insert(T item)
{
    if (elements.size() == 0)            // trivial case
    {   elements.add(item);
        return;
    }
    int pos = binarySearch(item);
    if (pos >= 0)
    {   System.out.println("Item already present");
        return;
    }
    else
        elements.add(-pos - 1);

}

//removes a specified item
public void remove(T item)
{
      int pos = binarySearch(item);
      if (pos<0)
      {
            System.out.println("No such element");
            return;
      }
      else
            elements.remove(pos);

}


}
```

```java
//Simple demo to illustrate why we return (-(mid+1)) and (-(mid+2)) in
//binary search
public class OrderedListDemo
{
      public static void main(String[] args)
      {
            OrderedList<String> names = new OrderedList<String>();
            names.insert("B");
            names.insert("C");
            names.insert("E");
            names.insert("G");
            names.enumerate();
            System.out.println("Search E:" + names.binarySearch("E"));
            System.out.println("Search F:" + names.binarySearch("F"));
            System.out.println("Search H:" + names.binarySearch("H"));
            System.out.println("Search A:" + names.binarySearch("A"));
      }


}
```

```java
//Simple orderedlist demo. Reads a text file of names and creates
//and prints the list.
import java.util.Scanner;
import java.io.*;
public class OrderedListDemo1
{
      public static void main(String[] args)throws IOException
      {
            Scanner keyboard = new Scanner(System.in);
            System.out.print("Enter the filename to read from: ");
            String filename = keyboard.nextLine();

            File file = new File(filename);
            Scanner inputFile = new Scanner(file);
            OrderedList<String> names = new OrderedList<String>();
            while(inputFile.hasNext())
            {
                  String s = inputFile.nextLine();
                  names.insert(s);
            }
            inputFile.close();
            names.enumerate();
      }


}
```

MERGING ORDERED LISTS

Merging two ordered lists and related operations are important.

Suppose the first ordered list L1 is as follows:

L1 (List 1)

Amar	Boris	Charlie	Dan	Fujian	Inder	Travis

↑f1

and the second ordered list L2 is as follows:

L2 (List 2)

Alex	Ben	Betty	Charlie	Dan	Pei	Travis	Zola	Zulu

↑f2

The merging of L1 and L2 should produce the following:

L3 (List 3)

Alex	Amar	Ben	Betty	Boris	Charlie	Dan	Fujian	Inder	Pei	Travis	Zola	Zulu

Two-finger-walking algorithm

```
Result list L3 ← empty
f1 ← 0   (start of L1)
f2 ← 0   (start of L2)
while ( f1 < length of L1 && f2 < length of L2)
{
    if (item at f1 < item at f2)
    {
        append item at f1 to L3 (add to end)
        move f1   (f1++)
    }
    else if (item at f2 < item at f1)
    {
        append item at f2 to L3
        move f2   (f2++)
    }
    else
    {
        append item at f1 to L3
        f1++
        f2++
    }
}
```

if (f1 == length of L1) append remaining items in L2 to L3

if (f2 == length of L2) append remaining items in L1 to L3