Outline

- Recursion
- Binary Search
- Exam II Review

Recursion

Recursion

- **What?** A recursive function is one that calls itself.
- **Why?** Break complex problems down to a smaller solvable problem.

Recursive Function

A recursive function must have two parts:

- **Base case(s)**
- **Recursive case(s)**

Recursion

```
void loop(){
  cout << "this is recursive function" << endl;

  loop();
}
```

What’s the problem with the function?

Motivation Example

How to compute $n!$ mathematically? say $5!$

$$5! = 5 * 4 * 3 * 2 * 1$$

Motivation Example

How to compute $n!$ mathematically? say $5!$

$$5! = 5 * 4 * 3 * 2 * 1$$

$n! = 1 * 2 * \cdots * n$ if $n > 0$.
$n! = 1$ if $n = 0$.

Recursion

```
long fact(int n) {

   if (n <= 1) return 1; // Base case

   return n * fact(n-1); // Recursive call
}
```

Another Recursion Example

In mathematics, the **Fibonacci series** can be defined as:

$F_0 = n,$ if $n \leq 1$

$F_n = F_{n-1} + F_{n-2},$ else $(n > 1)$

Another Recursion Example

In mathematics, the **Fibonacci series** can be defined as:

```
int fib ( int n ){

   if ( n <= 0 )
      return 0;
   else if ( n == 1 )
       return 1;
   else
      return fib( n - 1 ) +  fib( n - 2 );
}
```

Binary Search

Binary Search

```
int binarySearch (int A[], int key, int min, int max);
```

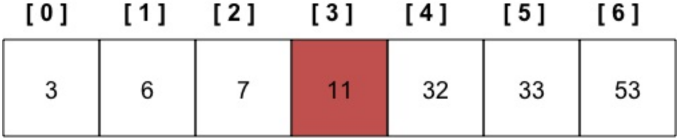
Binary Search – Without Recursion

Algorithm 1 $Iterative\text{-}Binary\text{-}Search(A, key, low, high)$

```
while $low \leq high$ do
    $mid = \lfloor (low + high)/2 \rfloor$
    if $key == A[mid]$ then
        return $mid$
    else if $key > A[mid]$ then
        $low = mid + 1$
    else
        $high = mid - 1$
return NIL
```

Binary Search – Example

key =7 low=0 high=6

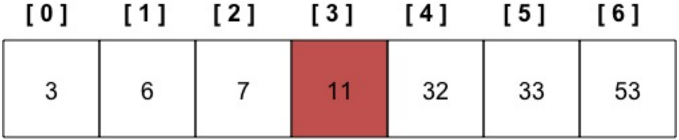


[0]	[1]	[2]	[3]	[4]	[5]	[6]
3	6	7	11	32	33	53

$low \leq high?$
$mid = \lfloor (low + high)/2 \rfloor = \lfloor (0+6)/2 \rfloor = 3$

Binary Search – Example

key =7 low=0 high=6



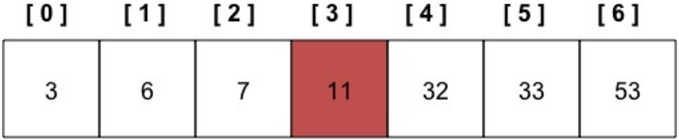
[0]	[1]	[2]	[3]	[4]	[5]	[6]
3	6	7	11	32	33	53

$low \leq high?$
$mid = \lfloor (low + high)/2 \rfloor = \lfloor (0+6)/2 \rfloor = 3$

Is $key == A[mid]?$

Binary Search – Example

key =7 low=0 high=6



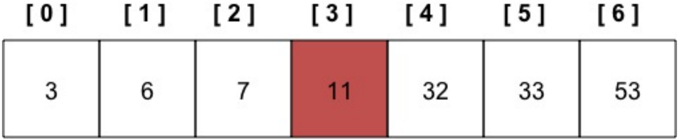
[0]	[1]	[2]	[3]	[4]	[5]	[6]
3	6	7	11	32	33	53

$low \leq high?$
$mid = \lfloor (low + high)/2 \rfloor = \lfloor (0+6)/2 \rfloor = 3$

Is $key > A[mid]?$

Binary Search – Example

key =7   low=0   high=6



[0]	[1]	[2]	[3]	[4]	[5]	[6]
3	6	7	11	32	33	53

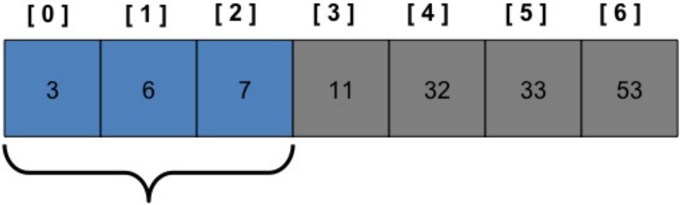
$low \leq high?$

$mid = \lfloor (low + high)/2 \rfloor = \lfloor (0+6)/2 \rfloor = 3$

Is $key < A[mid]?$

Binary Search – Example

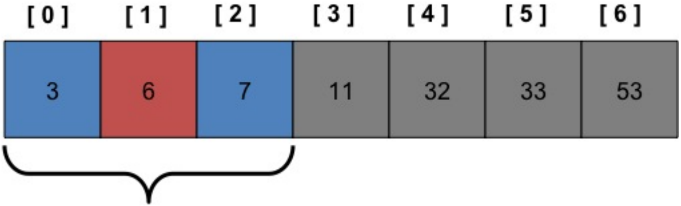
key =7 **low=0** **high=mid-1=2**



$low \leq high?$

Binary Search – Example

key =7 **low=0** **high=2**



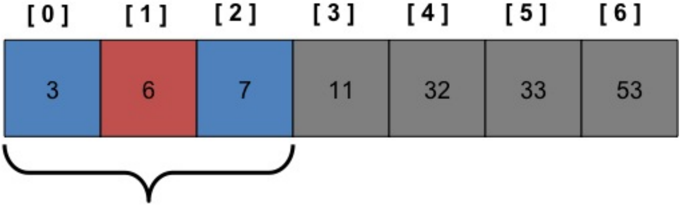
[0]	[1]	[2]	[3]	[4]	[5]	[6]
3	6	7	11	32	33	53

$low \leq high?$

$mid = \lfloor (low + high)/2 \rfloor = \lfloor (0 + 2)/2 \rfloor = 1$

Binary Search – Example

key =7 low=0 high=2



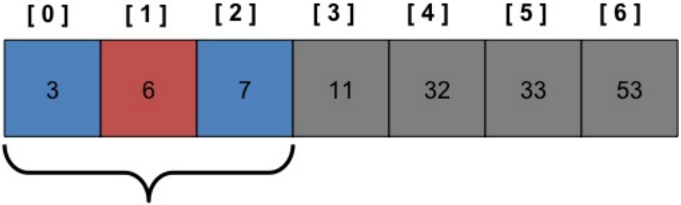
[0]	[1]	[2]	[3]	[4]	[5]	[6]
3	6	7	11	32	33	53

$low \leq high?$
$mid = \lfloor (low + high)/2 \rfloor = \lfloor (0 + 2)/2 \rfloor = 1$

Is $key == A[mid]?$

Binary Search – Example

key =7 **low=0** **high=2**



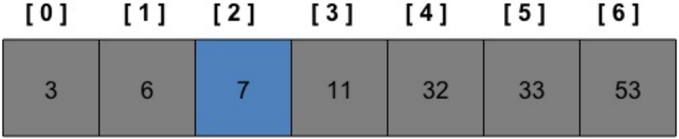
[0]	[1]	[2]	[3]	[4]	[5]	[6]
3	6	7	11	32	33	53

$low \leq high?$
$mid = \lfloor (low + high)/2 \rfloor = \lfloor (0+2)/2 \rfloor = 1$

Is $key > A[mid]?$

Binary Search – Example

key =7 **low=mid+1=2** **high=2**

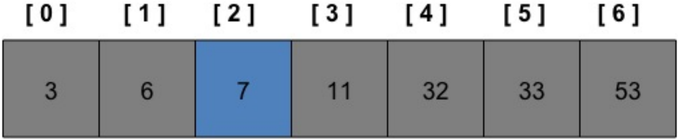


[0]	[1]	[2]	[3]	[4]	[5]	[6]
3	6	7	11	32	33	53

$low \leq high?$
$mid = \lfloor (low + high)/2 \rfloor = \lfloor (2 + 2)/2 \rfloor = 2$

Binary Search – Example

key =7 **low=mid+1=2** **high=2**



[0]	[1]	[2]	[3]	[4]	[5]	[6]
3	6	7	11	32	33	53

$low \leq high?$
$mid = \lfloor (low + high)/2 \rfloor = \lfloor (2 + 2)/2 \rfloor = 2$

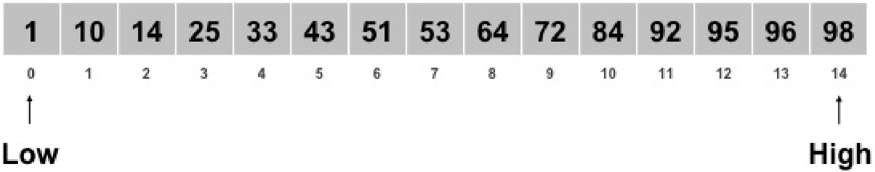
Is $key == A[mid]?$

Analysis of Binary Search

Given $n = 7$, how many times has the body of **while** loop body been executed?

Binary Search – Another Example

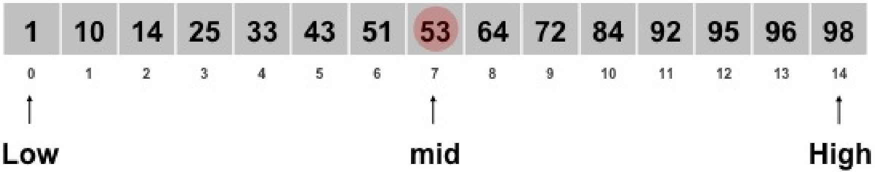
key = 33



1	10	14	25	33	43	51	53	64	72	84	92	95	96	98
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

Binary Search – Another Example

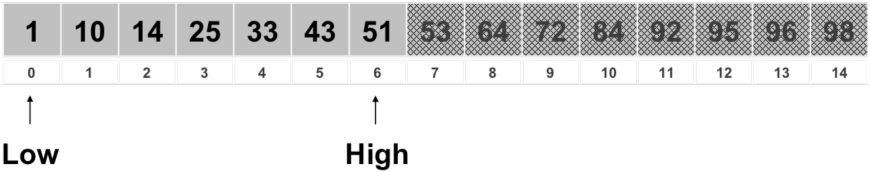
key = 33



1	10	14	25	33	43	51	53	64	72	84	92	95	96	98
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

Binary Search – Another Example

key = 33



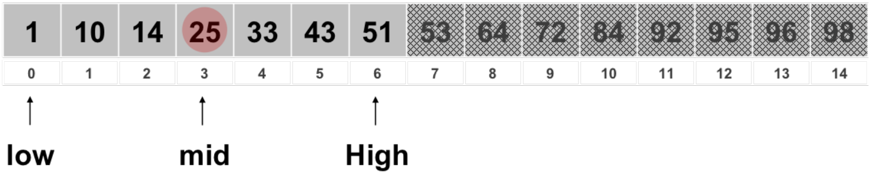
1	10	14	25	33	43	51	53	64	72	84	92	95	96	98
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

Low

High

Binary Search – Another Example

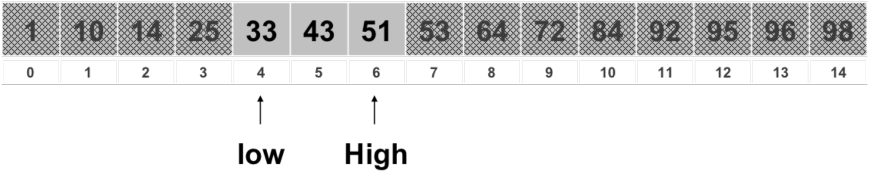
key = 33



1	10	14	25	33	43	51	53	64	72	84	92	95	96	98
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

Binary Search – Another Example

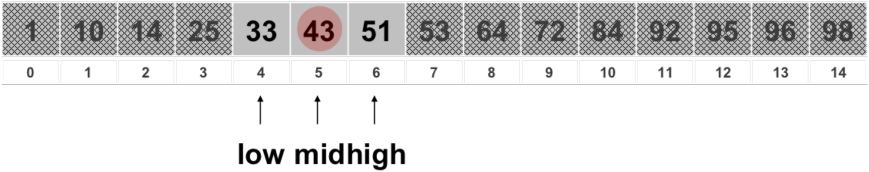
key = 33



1	10	14	25	33	43	51	53	64	72	84	92	95	96	98
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

Binary Search – Another Example

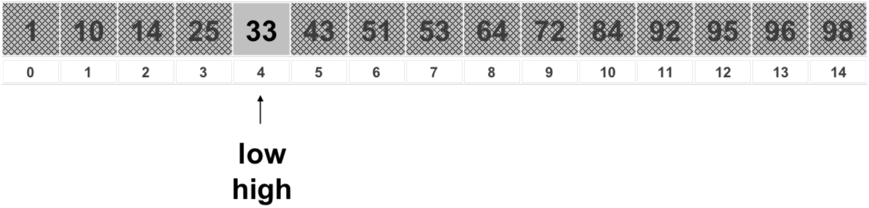
key = 33



1	10	14	25	33	43	51	53	64	72	84	92	95	96	98
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

Binary Search – Another Example

key = 33

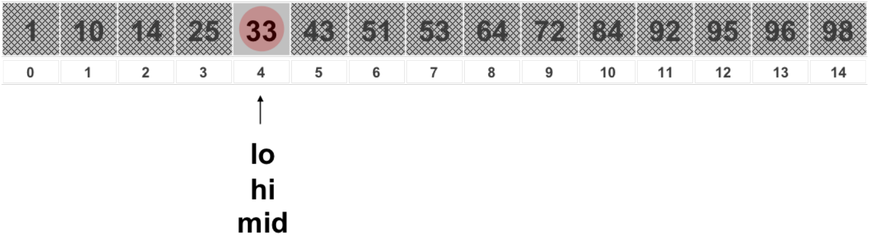


1	10	14	25	33	43	51	53	64	72	84	92	95	96	98
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

low
high

Binary Search – Another Example

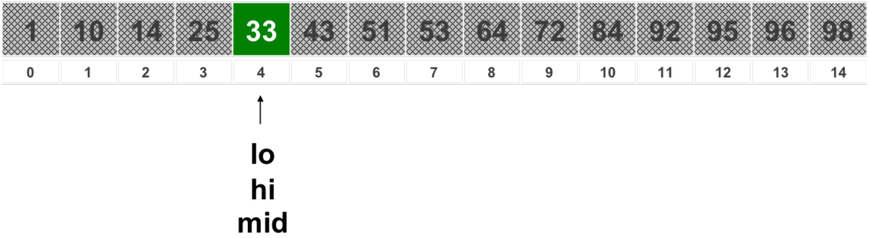
key = 33



1	10	14	25	33	43	51	53	64	72	84	92	95	96	98
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

Binary Search – Another Example

key = 33



1	10	14	25	33	43	51	53	64	72	84	92	95	96	98
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

Binary Search – Example Implementation

```
int binarySearch(int list[], int key, int low, int high){

   while(low <= high){
     int mid = (low+high)/2;
     if(key == list[mid])
        return mid;
     else if (key>list[mid])
        low = mid+1;
     else
        high = mid-1;
   }
   return -1;
}
```

Binary Search – Recursive Algorithm

Algorithm 2 $Recursive\text{-}Binary\text{-}Search(A, key, low, high)$

```
if $low > high$ then
    return NIL
$mid = \lfloor (low + high)/2 \rfloor$
if $key == A[mid]$ then
    return $mid$
else if $key > A[mid]$ then
    return $Recursive\text{-}Binary\text{-}Search(A, key, mid + 1, high)$
else
    return $Recursive\text{-}Binary\text{-}Search(A, key, low, mid - 1)$
```

Exam II Review

Exam II

Closed Book, closed notes, no cellphone.

Study Resources

- Textbook

Study Resources

- Textbook
- Lecture Notes

Study Resources

- Textbook
- Lecture Notes
- Homeworks 4, 5 and Quiz 2.

Main Topics

- C++ Class/Objects
- Dynamic Array
- Linked List
- Stack
- Queue
- Template

C++ Class

- Constructor/Copy Constructor/Default Constructor
- Destructor
- Member Function/Member Variables
- Friend Function /Nonmember Function
- Operator Overloading
- Access Control Specifiers/Modifiers

Linked List Data Structure

- What is the linked list data structure

Linked List Data Structure

- What is the linked list data structure
- What are the essential member variables

Linked List Data Structure

- What is the linked list data structure
- What are the essential member variables
- What are the basic operations?

Linked List Data Structure

- What is the linked list data structure
- What are the essential member variables
- What are the basic operations?
- How to traverse the linked list?

Practice Questions

```
class List {
   private:
     struct Node{
        float value;
        Node *next;
     }
     List head;

public:
   ...
   void addFront(const float& f);
}
```

Practice Questions – addFront

```
void List :: addFront(const float& f){


}
```

Practice Questions – addFront

```
void List :: addFront(const float& f){
  Node *v = new Node;
  v->value = f;
  v->next = head;
  head = v;
}
```

Practice Questions – removeFront

```
class List {
   private:
     struct Node{
        float value;
        Node *next;
     }
     List head;

public:
   ...
   void addFront(const float& f);
   void removeFront();

}
```

Practice Questions – removeFront

```
void List :: removeFront(){



}
```

Practice Questions – removeFront

```
void List :: removeFront(){
  Node *temp = head;
  head = temp ->next;
  delete temp;
}
```

Practice Questions – removeFront

```
class List {
   private:
     struct Node{
        float value;
        Node *next;
     }
     List head;

public:
   ...
   void addFront(const float& f);
   void removeFront();
   float getFront();

}
```

Practice Questions – getFront

```
float List :: getFront(){

}
```

Practice Questions – getFront

```
float List :: getFront(){
  return head->value;
}
```

Practice Questions – operator overloading

```
class List {
   private:
     struct Node{
        float value;
        Node *next;
     }
     List head;

public:
   ...
   void addFront(const float& f);
   void removeFront();
   float getFront();
   friend List operator+(List l1, List l2);

}
```

Practice Questions – Integer class

```
class List {
   private:
     int data;

public:
   // constructor

   // basic member functions


}
```

Practice Questions(Cont')

Given the following definition of a singly list header file, write a **recursive delete** method for the lists with integer data that deletes the first occurrence of a given integer from the list and returns the resulting list.

```
class List{
   struct Node{
      int value;
      Node *next;
   }
   public:
    List *head;
    List(){head = NULL;}
    List appendNode(int i);
   //...
}
```

Practice Questions(Cont')

Given the following definition of a singly list header file, write a **recursive delete** method for the lists with integer data that deletes the first occurrence of a given integer from the list and returns the resulting list.

```
// pre: none
//post: delete the first occurrence of i in list r
List :: List delete(int i, List *r) {




}
```