APS105
Winter 2012

Jonathan Deber
jdeber -at- cs -dot- toronto -dot- edu

Lecture 31
April 2, 2012

Today

- `char` Functions
- Searching

- A5 Extension

char Functions

`char` Functions

- C library has functions for working with `char`s
- `#include <ctype.h>` (no `-l` needed)
- These are not string functions!

ctype.h

- Classification Functions
- Conversion Functions

Classification

- `isalpha()`
- `isdigit()`
- `isupper()`
- `islower()`
- `isspace()`
- *etc.*

```
isdigit('1')    true

isupper('a')    false
```

~~`bool islower(char c);`~~

`int islower(int c);`

Predates `stdbool.h`

Needs to be able to handle abnormal situations

~~`char getchar(void);`~~

`int getchar(void);`

It can return one of `256` possible `char`s (`ASCII` codes `0`-`255`) or a special `char` `EOF` meaning "end of file"

That's 257 possible values!

```
bool islower(char c);
```

99.9% of time, you can assume this

```
int islower(int c);
```

Predates `stdbool.h`

Needs to be able to handle abnormal situations

~~`char getchar(void);`~~

```
int getchar(void);
```

It can return one of 256 possible `char`s (`ASCII` codes `0`-`255`) or a special `char` `EOF` meaning "end of file"

That's 257 possible values!

```
bool isStringLower(char *s)
{
    bool result = true;

    for (int i = 0; s[i] != '\0'; i++)
    {
       if (! islower(s[i]))
       {
           result = false;
       }
    }
    return result;
}
```

```
bool isStringLower(char *s)
{
    bool result = true;

    for (int i = 0; s[i] != '\0'; i++)
    {
        result = result && islower(s[i]);



    }
    return result;
}
```

```
isStringLower("ABC")    false
isStringLower("aBc")    false
isStringLower("123")    false
isStringLower("abc")    true
```

Conversion

- `toupper()`
- `tolower()`
- Safer version of `c ± 32`

```
char c;
c = toupper('a');
c = toupper('A');
c = toupper('3');
```

`'A'`
`'A'`
`'3'`

Conversion Prototypes

```
char toupper(char c);
```

```
int toupper(int c);
```

Conversion Prototypes

```
char toupper(char c);
```

99.9% of time, you can assume this

```
int toupper(int c);
```

```
void stringToUpper(char *s)
{
    for (int i = 0; s[i] != '\0';i++)
    {
        s[i] = toupper(s[i]);
    }
}
```

```
stringToUpper("abc")
```

Wrong

```
char s1[] = "ABC";
stringToUpper(s1)
```

ABC

```
char s2[] = "aBc";
stringToUpper(s2)
```

ABC

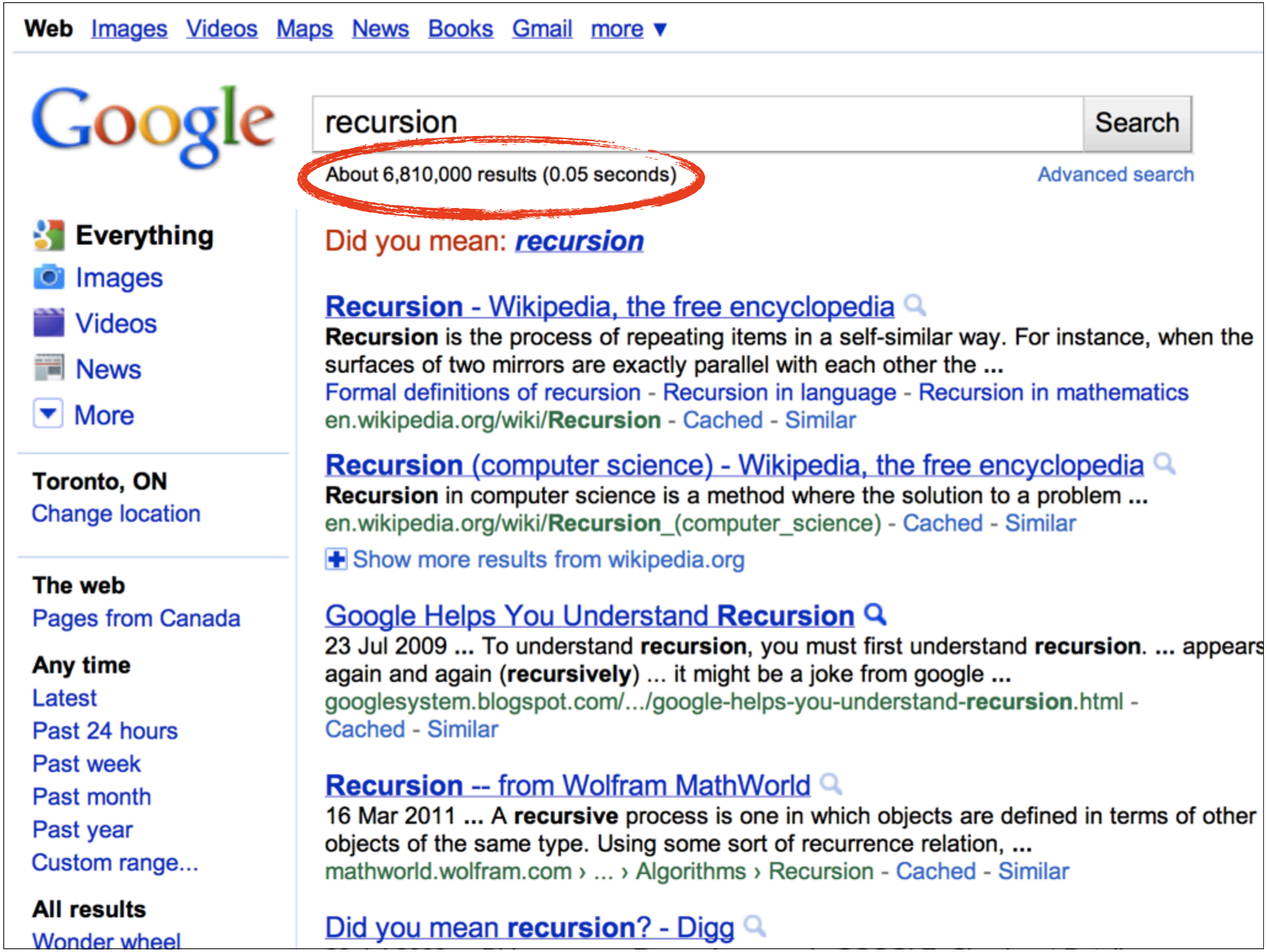
```
char s3[] = "aBc12";
stringToUpper(s3)
```

ABC12

Searching

Searching

- Given a collection of data and an item
  - Find the location of that item
  - Or figure out it isn't there


Web Images Videos Maps News Books Gmail more ▼
Google
recursion
Search
About 6,810,000 results (0.05 seconds)
Advanced search
Everything
Images
Videos
News
More
Toronto, ON
Change location
The web
Pages from Canada
Any time
Latest
Past 24 hours
Past week
Past month
Past year
Custom range...
All results
Wonder wheel
Did you mean: recursion
Recursion - Wikipedia, the free encyclopedia
Recursion is the process of repeating items in a self-similar way. For instance, when the surfaces of two mirrors are exactly parallel with each other the ...
Formal definitions of recursion - Recursion in language - Recursion in mathematics
en.wikipedia.org/wiki/Recursion - Cached - Similar
Recursion (computer science) - Wikipedia, the free encyclopedia
Recursion in computer science is a method where the solution to a problem ...
en.wikipedia.org/wiki/Recursion_(computer_science) - Cached - Similar
Show more results from wikipedia.org
Google Helps You Understand Recursion
23 Jul 2009 ... To understand recursion, you must first understand recursion. ... appears again and again (recursively) ... it might be a joke from google ...
googlesystem.blogspot.com/.../google-helps-you-understand-recursion.html - Cached - Similar
Recursion -- from Wolfram MathWorld
16 Mar 2011 ... A recursive process is one in which objects are defined in terms of other objects of the same type. Using some sort of recurrence relation, ...
mathworld.wolfram.com › ... › Algorithms › Recursion - Cached - Similar
Did you mean recursion? - Digg

Searching

- Arrays vs. Linked Lists
- Unsorted vs. sorted collections of data

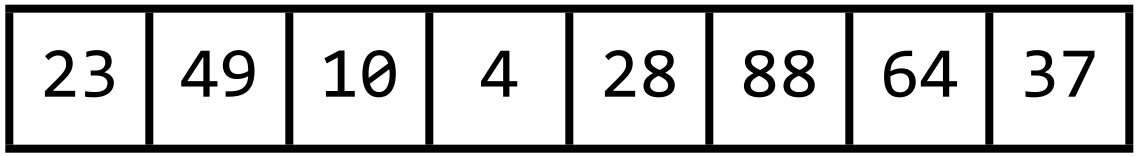
If it Isn't There

- Need to make sure you know how to indicate that you failed to find something
  - If you're returning indices, `-1` is common
  - If you're returning pointers, `NULL` is common
- These only work if those are not valid values!

Unsorted

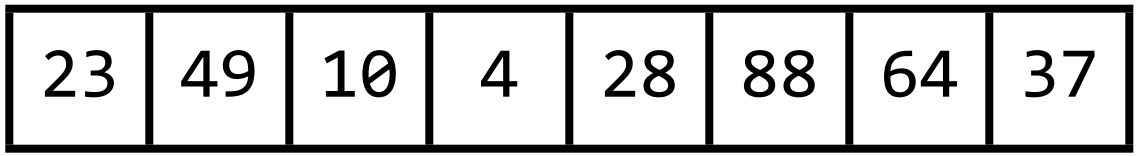
- Start at the beginning, keep looking
- By definition, you might have to examine the entire set of data

Unsorted Array



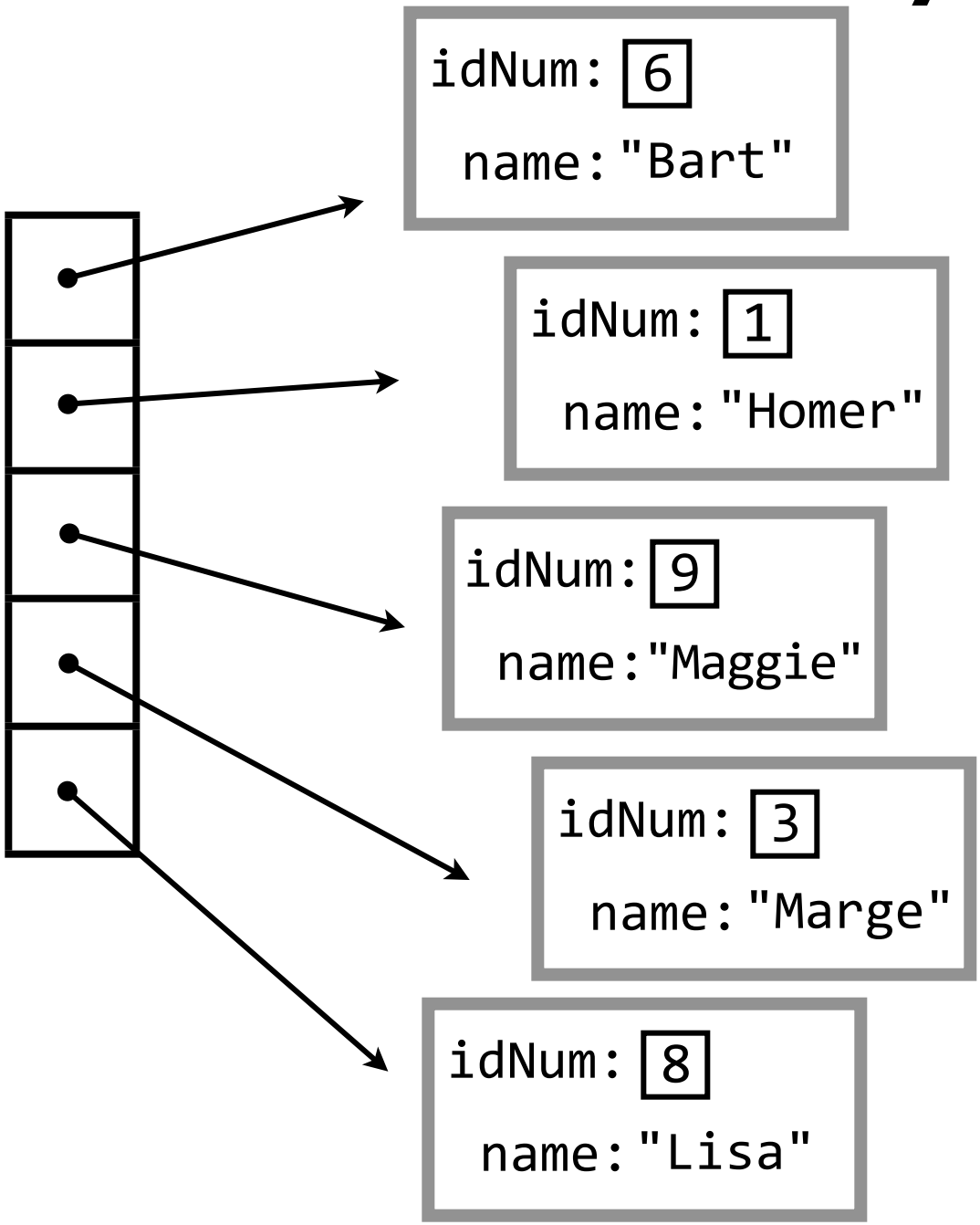

```
int findIndex(int list[], int n, int x)
{
    int index = -1;
    for (int i = 0; i < n; i++)
    {
        if (list[i] == x)
        {
            index = i;
        }
    }

    return index;
}
```

Unsorted Array

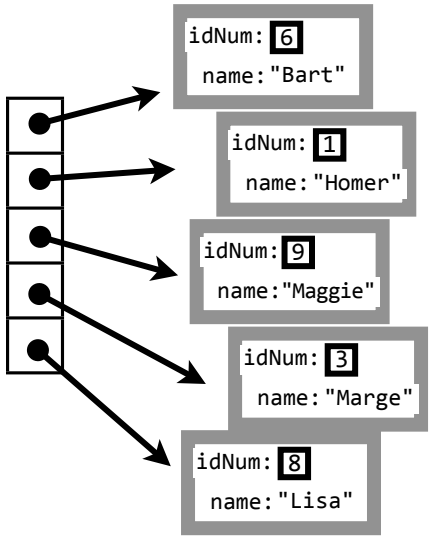



```
int findIndex(int list[], int n, int x)
{
    int index = -1;
    for (int i = 0; i < n && index == -1; i++)
    {
        if (list[i] == x)
        {
            index = i;
        }
    }

    return index;
}
```

Unsorted Array

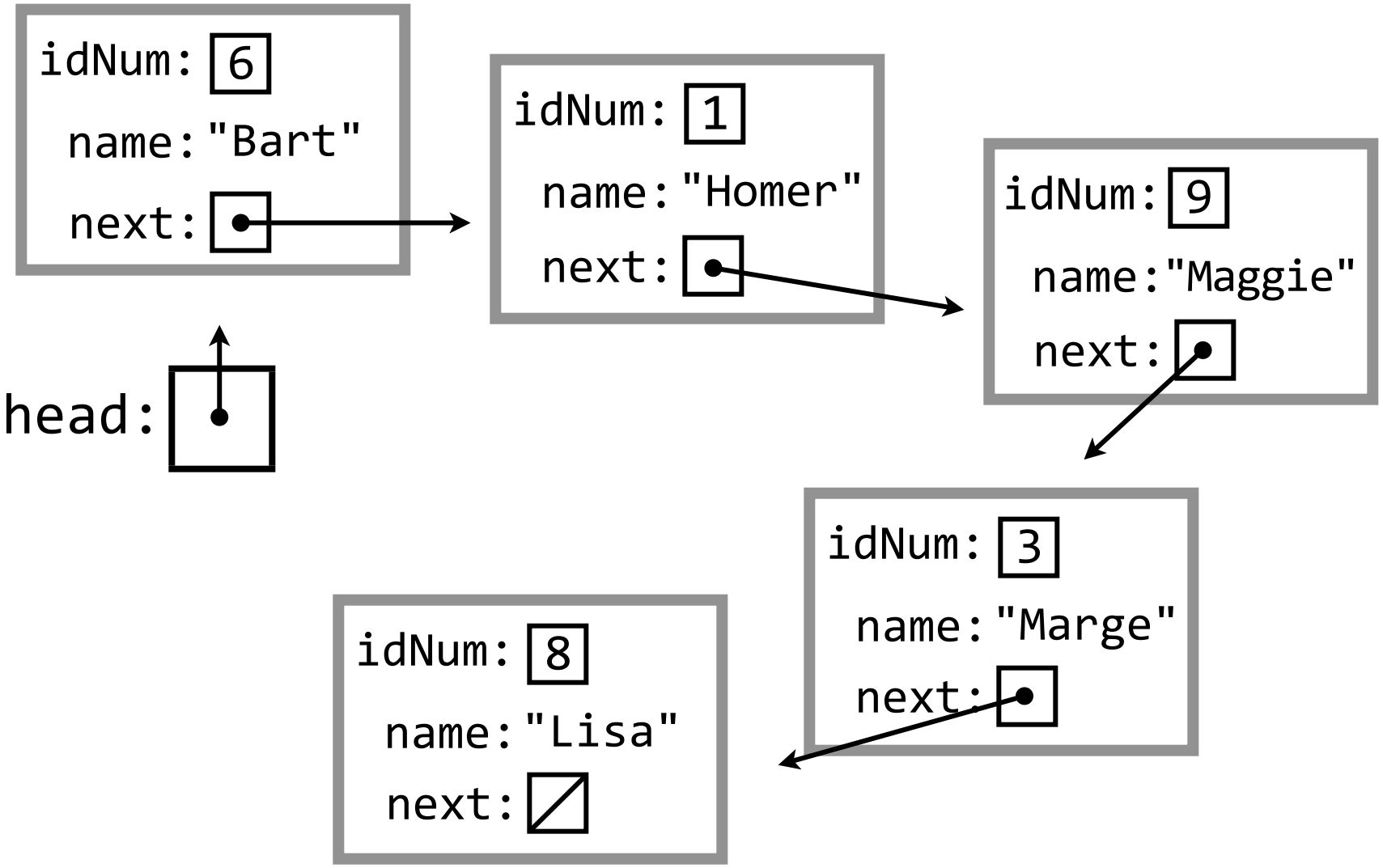


Unsorted Array




```
char *findName(Person *list[], int n, int idNum)
{
    char *result = NULL;
    for (int i = 0; i < n && result == NULL; i++)
    {
        if (list[i]->idNum == idNum)
        {
            result = list[i]->name;
        }
    }

    return result;
}
```

Unsorted Linked List



Unsorted Linked List

```
char *findName(Person *head, int idNum)
{
  char *result = NULL;

  Person *current = head;
  while (current != NULL)
  {
    if (current->idNum == idNum)
    {
      result = current->name;
    }
    current = current->next;
  }

  return result;
}
```

Unsorted Linked List

```
char *findName(Person *head, int idNum)
{
  char *result = NULL;

  Person *current = head;
  while (current != NULL && result == NULL)
  {
    if (current->idNum == idNum)
    {
      result = current->name;
    }
    current = current->next;
  }

  return result;
}
```

Unsorted

- Best case: it's the first item we look at
  - Only examine one item
- Worst case: it's the last item, or it's not in the list
  - Examine the entire list

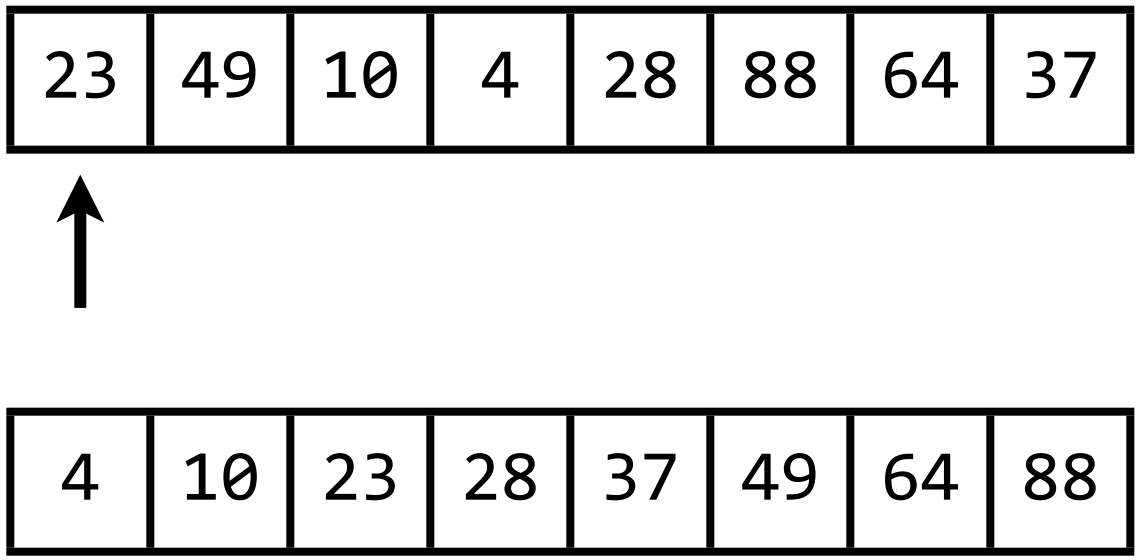
Sorted

- Items are in some known order
  - Numeric, alphabetical, etc.

23	49	10	4	28	88	64	37

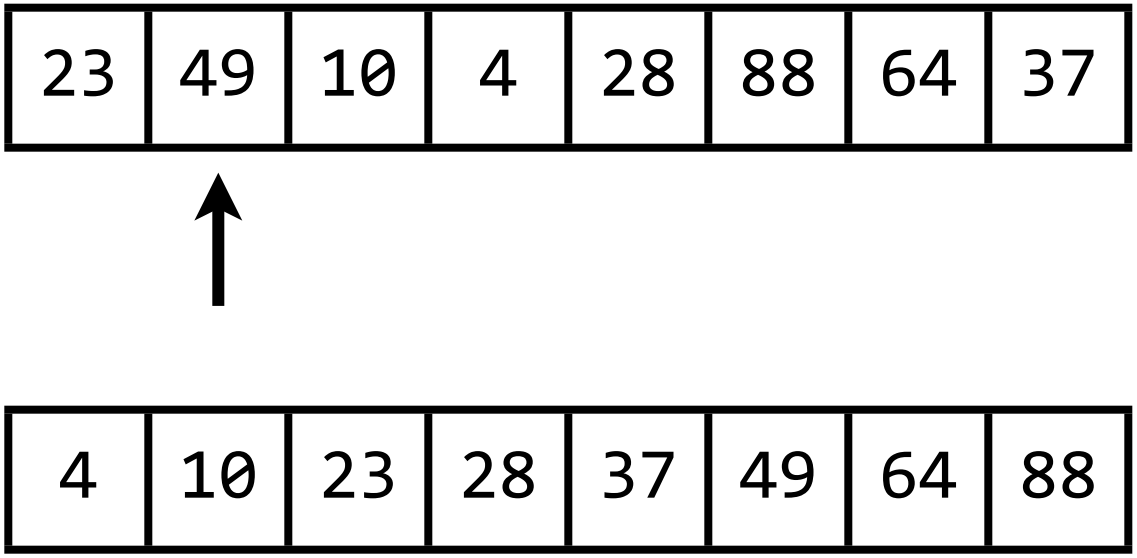
4	10	23	28	37	49	64	88

Unsorted vs. Sorted



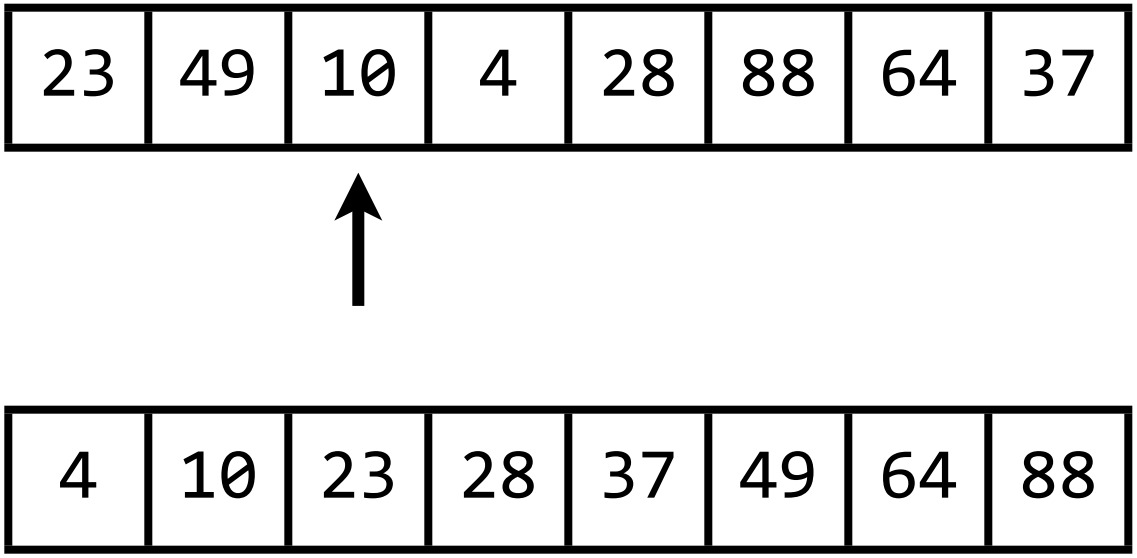
Looking for 35

Unsorted vs. Sorted



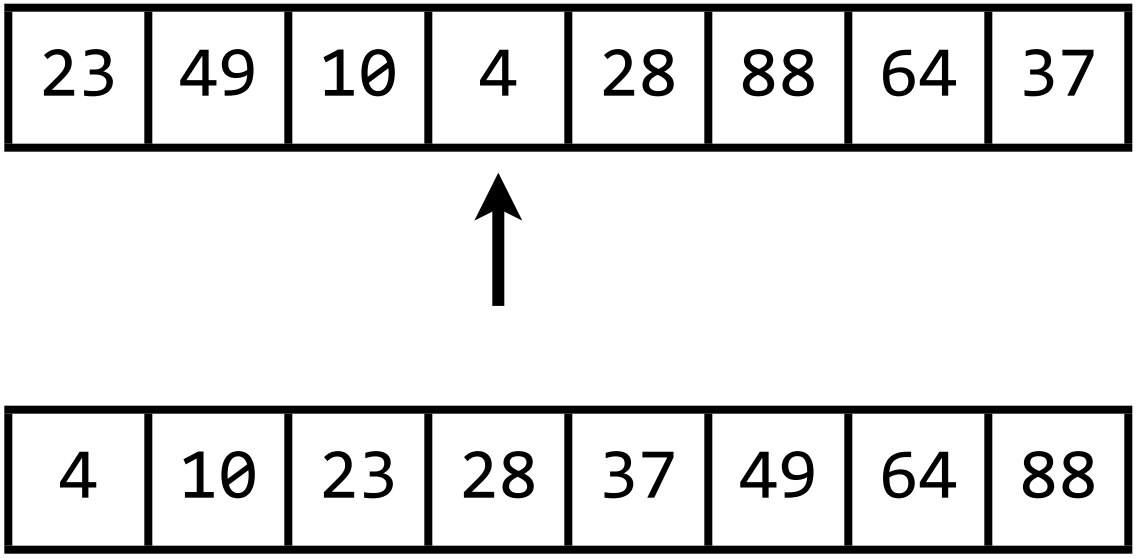
Looking for 35

Unsorted vs. Sorted



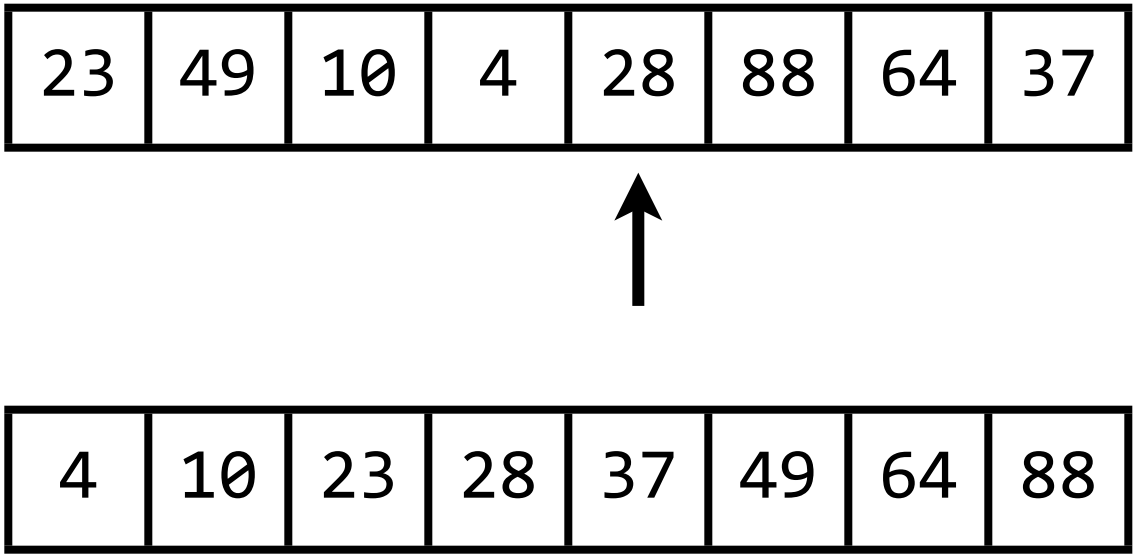
Looking for 35

Unsorted vs. Sorted



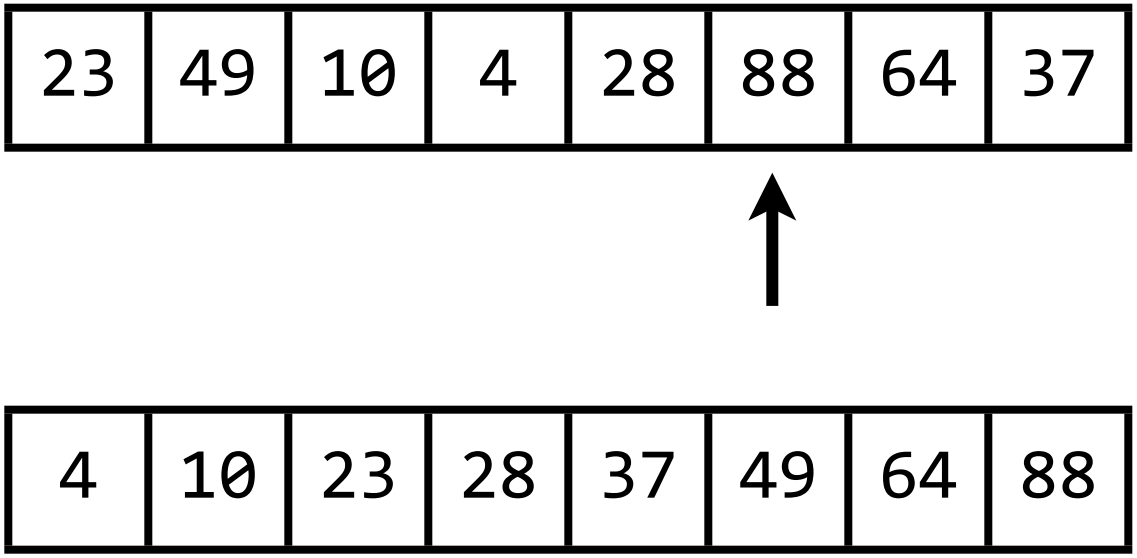
Looking for 35

Unsorted vs. Sorted



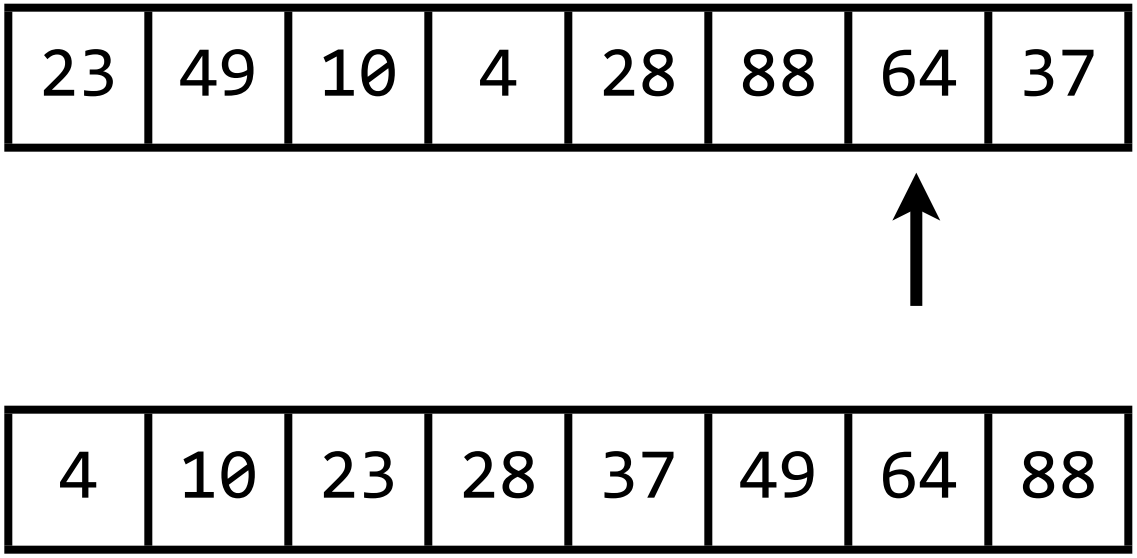
Looking for 35

Unsorted vs. Sorted



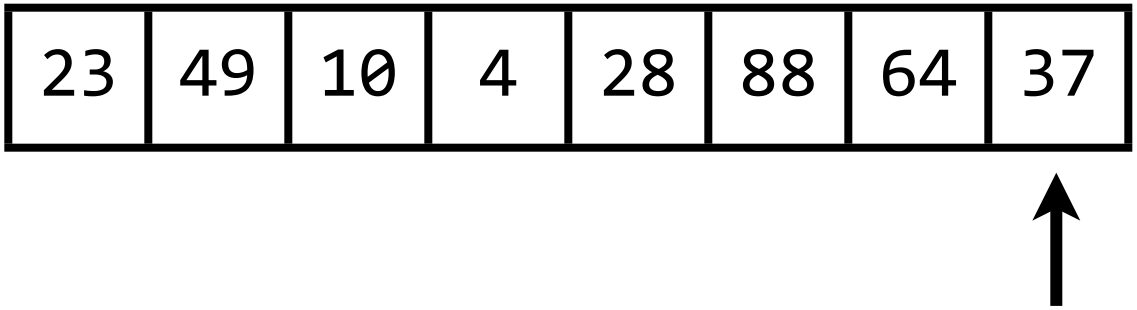
Looking for 35

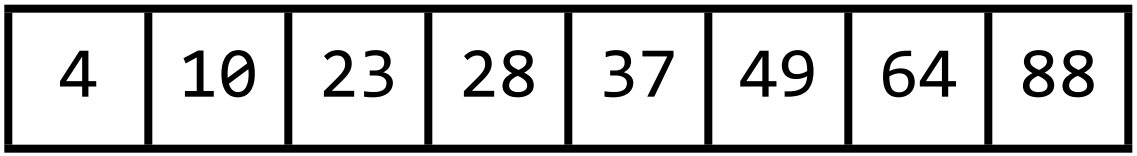
Unsorted vs. Sorted



Looking for 35

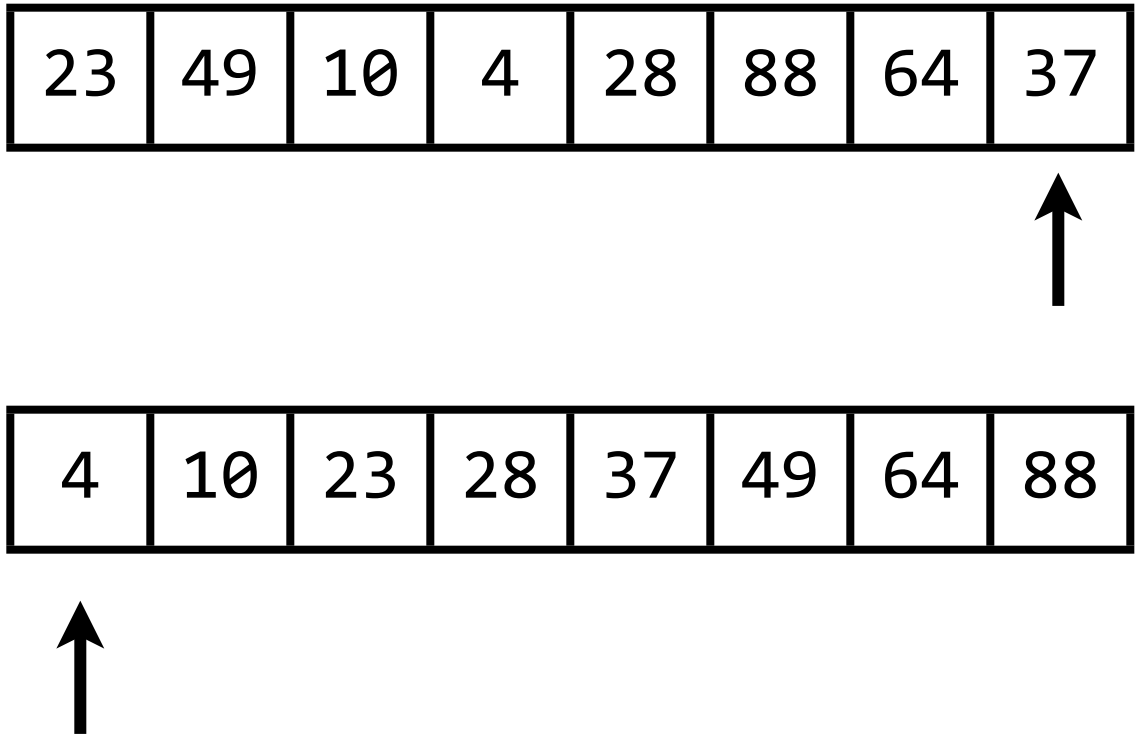
Unsorted vs. Sorted





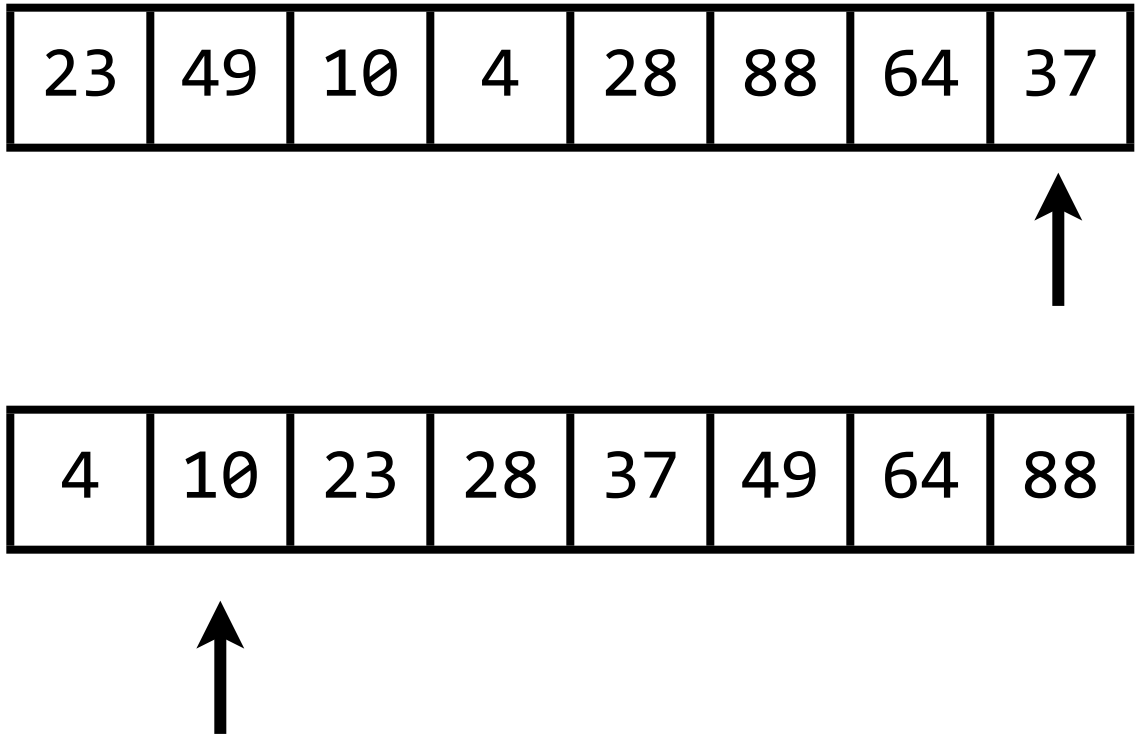
Looking for 35

Unsorted vs. Sorted



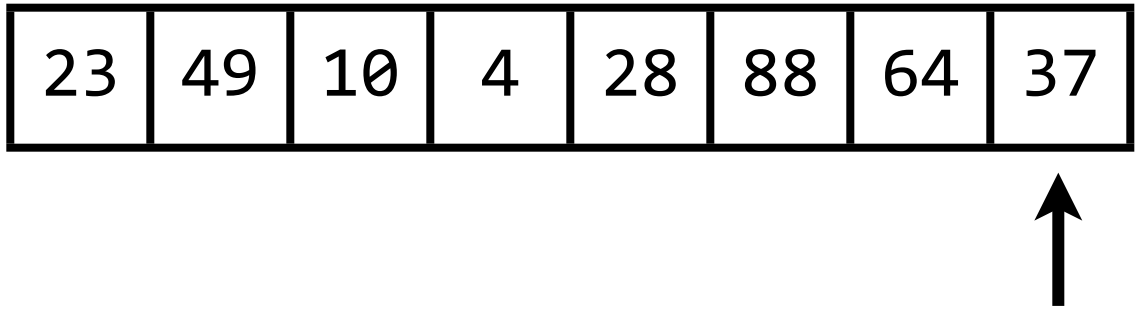
Looking for 35

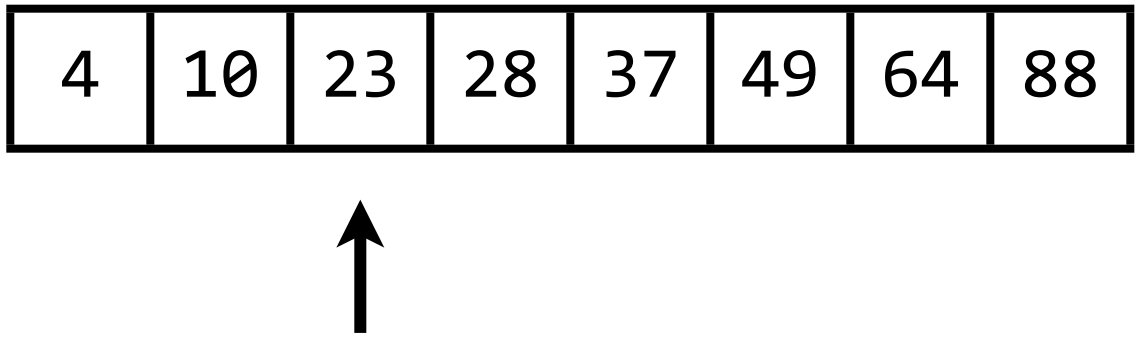
Unsorted vs. Sorted



Looking for 35

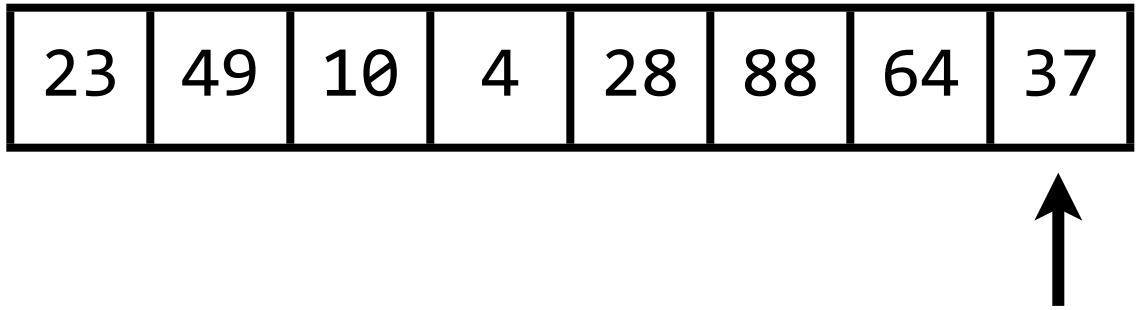
Unsorted vs. Sorted

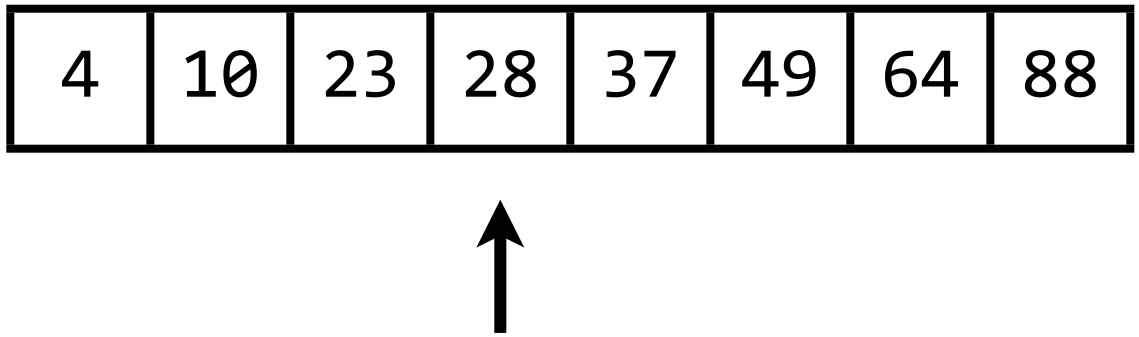




Looking for 35

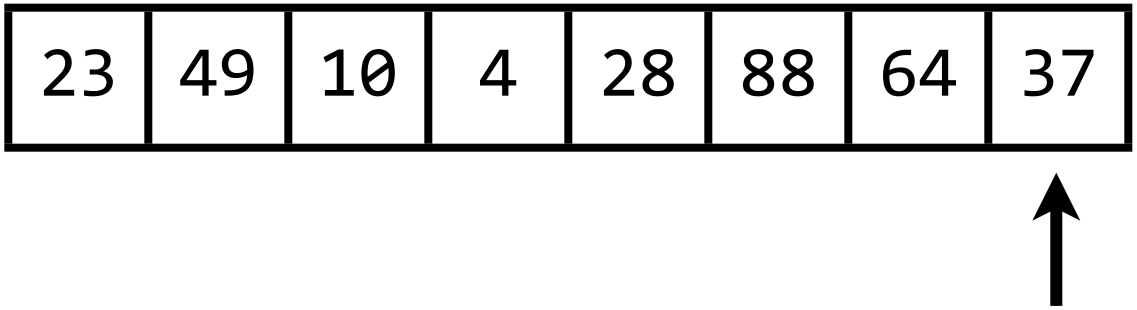
Unsorted vs. Sorted

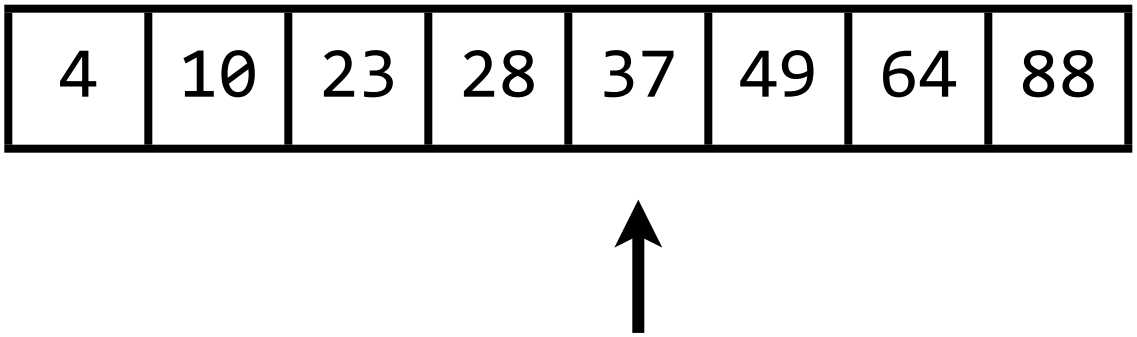




Looking for 35

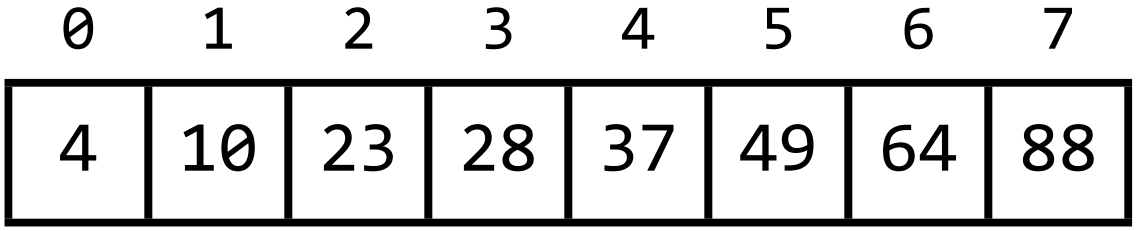
Unsorted vs. Sorted





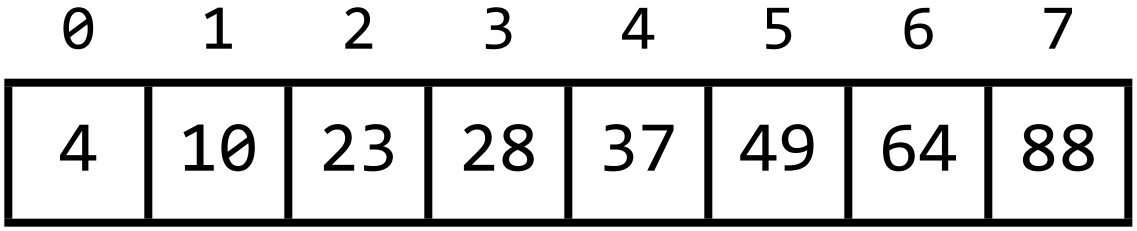
Looking for 35

Sorted Array



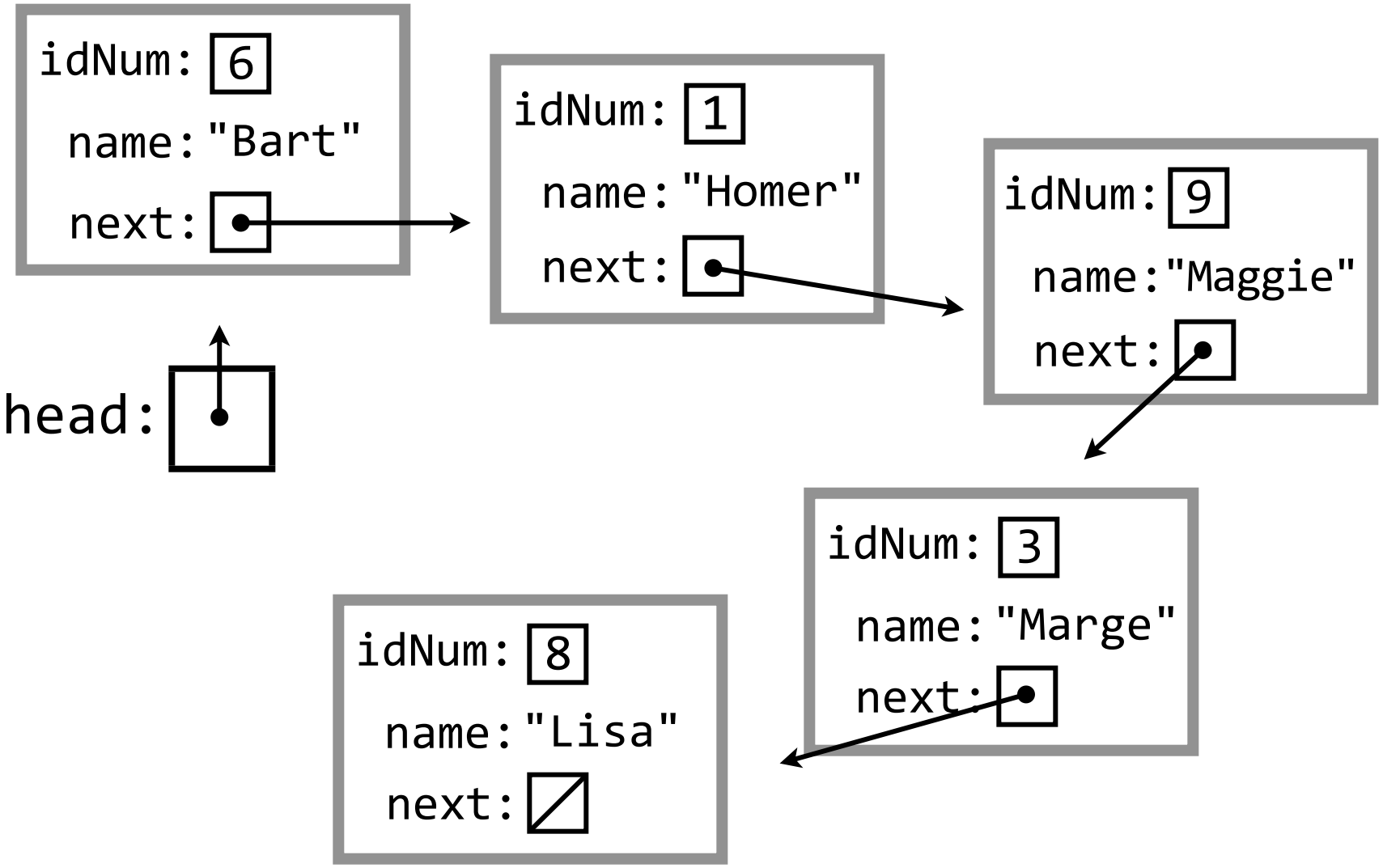

```
int findIndex(int list[], int n, int x)
{
  int index = -1;
  for (int i = 0; i < n && index == -1; i++)
  {
    if (list[i] == x)
    {
      index = i;
    }
  }

  return index;
}
```

Sorted Array

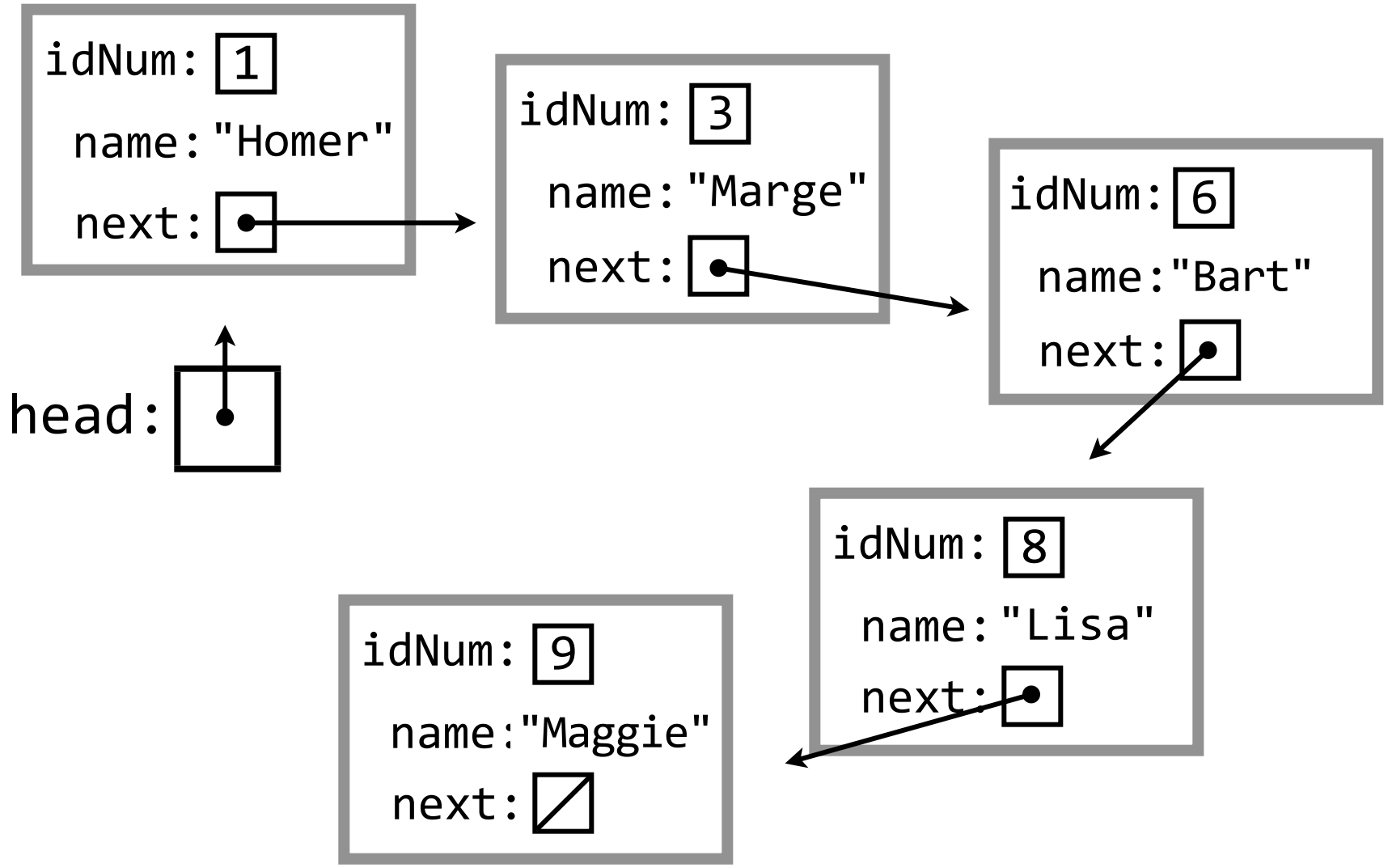



```
int findIndex(int list[], int n, int x)
{
  int index = -1;
  for (int i = 0; i < n && index == -1
                          && list[i] <= x; i++)
  {
    if (list[i] == x)
    {
      index = i;
    }
  }

  return index;
}
```

Sorted Linked List



Sorted Linked List



Sorted Linked List

```
char *findName(Person *head, int idNum)
{
  char *result = NULL;

  Person *current = head;
  while (current != NULL && result == NULL)
  {
    if (current->idNum == idNum)
    {
      result = current->name;
    }
    current = current->next;
  }

  return result;
}
```

Sorted Linked List

```
char *findName(Person *head, int idNum)
{
  char *result = NULL;

  Person *current = head;
  while (current != NULL && result == NULL
                         && current->idNum <= idNum)
  {
    if (current->idNum == idNum)
    {
      result = current->name;
    }
    current = current->next;
  }

  return result;
}
```

Sorted

- Might allow us to stop early

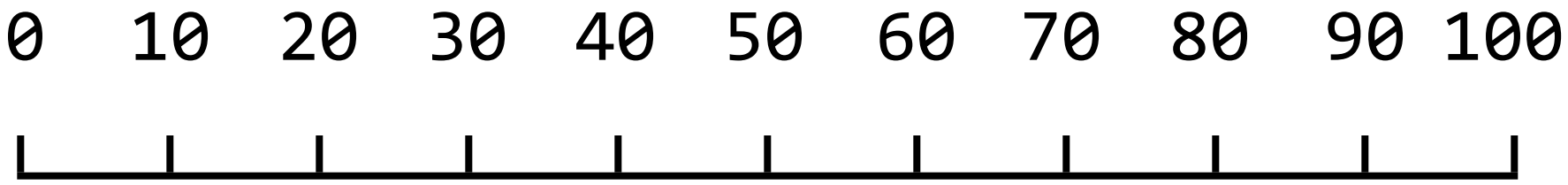
- Best case (same): it's the first item
- Worst case (same): it's the last item
- Variable case (new): it's not in the list
  - Might be able to stop after the first item, or might need to examine the entire list

Smarter Searching

- This is not taking full advantage of the sorted list
- We can search more quickly if we do

Guessing Game

- I'm thinking of a number between 0 and 100
- You guess a number, I'll tell you if my number is larger or smaller



Guessing Game

- I'm thinking of a number between 0 and 100
- You guess a number, I'll tell you if my number is larger or smaller

50

0 10 20 30 40 50 60 70 80 90 100

Guessing Game

- I'm thinking of a number between 0 and 100
- You guess a number, I'll tell you if my number is larger or smaller

50 75

0 10 20 30 40 50 60 70 80 90 100

Guessing Game

- I'm thinking of a number between 0 and 100
- You guess a number, I'll tell you if my number is larger or smaller

```
                    50   75   87

0   10  20  30  40  50  60  70  80  90 100
                                 ┴
                                ─┴─
```

Guessing Game

- I'm thinking of a number between 0 and 100
- You guess a number, I'll tell you if my number is larger or smaller

```
                          50   75   87   81

0   10  20  30  40  50  60  70  80  90 100
                                  ——
```

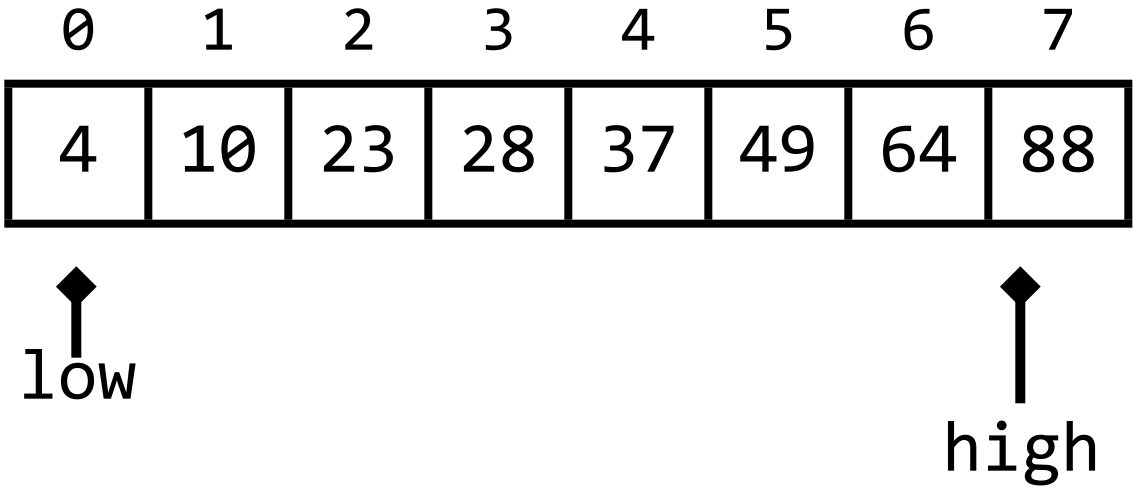
Guessing Game

- I'm thinking of a number between 0 and 100
- You guess a number, I'll tell you if my number is larger or smaller

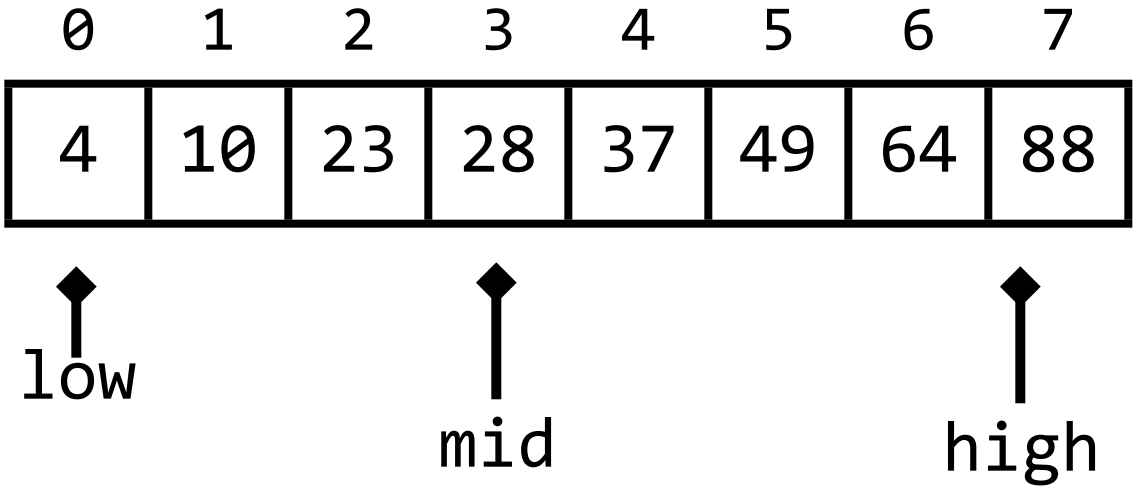
```
                                  50   75   87   81   84

0   10  20  30  40  50  60  70  80  90 100
                                  ——
```

Binary Search

- This is an algorithm called binary search
- Each iteration cuts the search space in half
- Only works on sorted lists

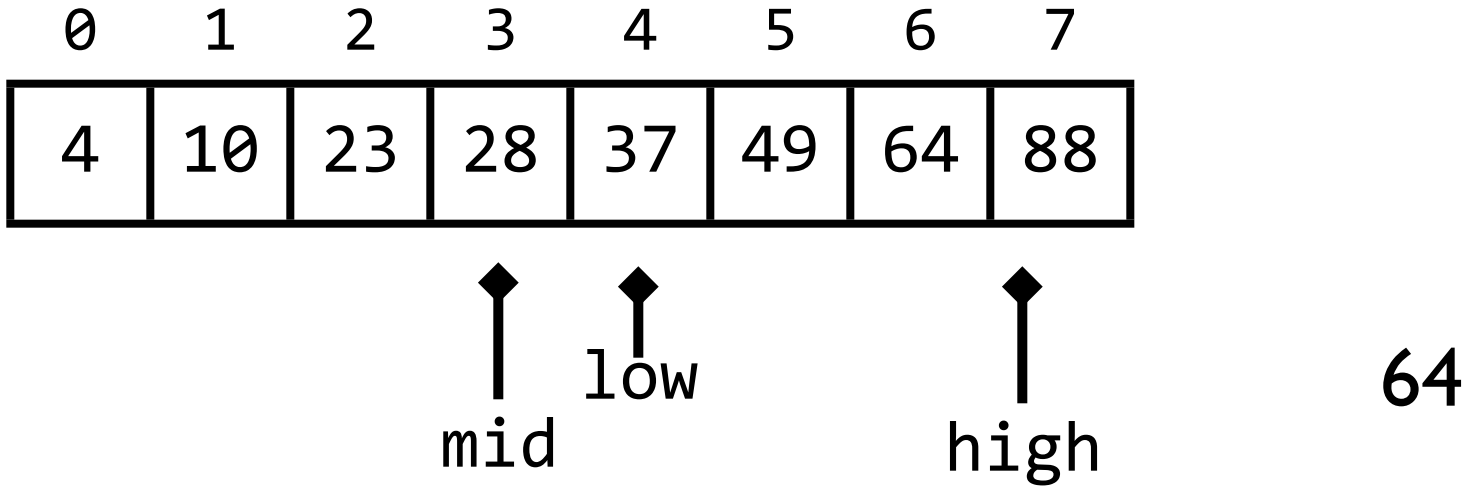


Find the middle of the list.
`if` (that item is larger than the item we're searching for)
{
  Binary search the list from `low` to `mid-1`
}
`else if` (that item is smaller than the item we're searching for)
{
  Binary search the list from `mid+1` to `high`
}
`else`
{
  `return mid`
}

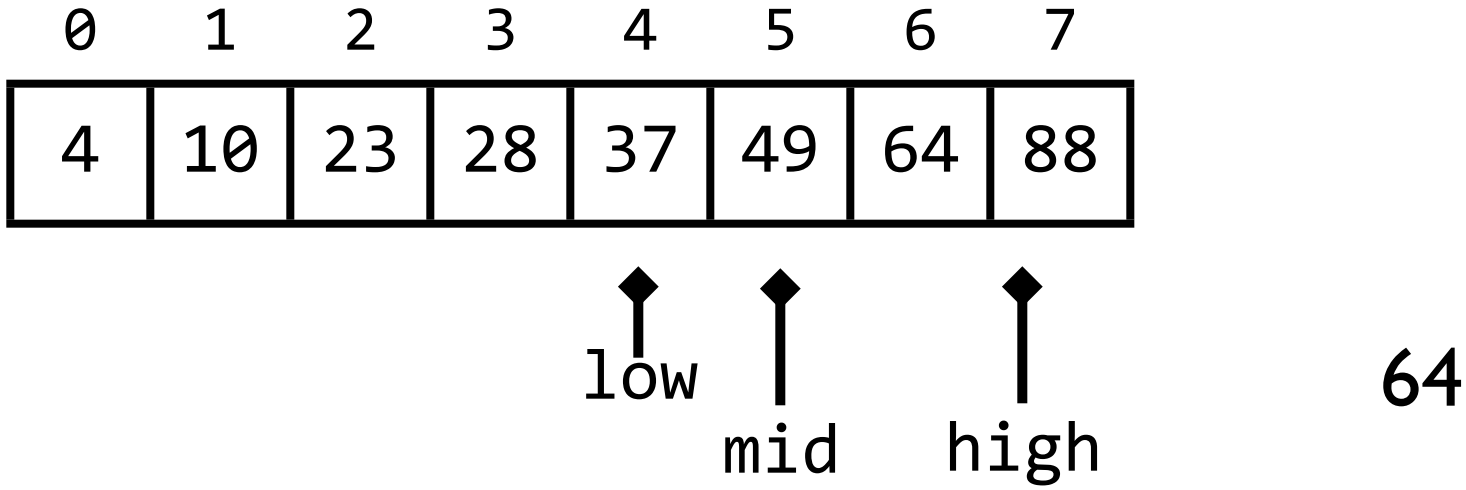


64

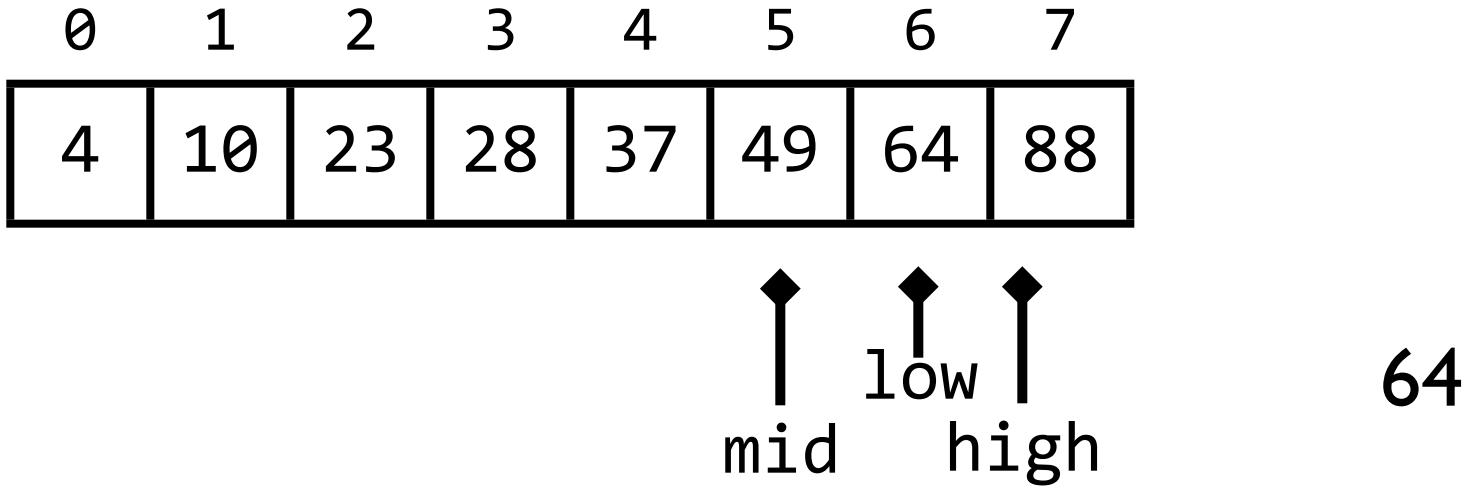
Find the middle of the list.
`if` (that item is larger than the item we're searching for)
{
  Binary search the list from `low` to `mid-1`
}
`else if` (that item is smaller than the item we're searching for)
{
  Binary search the list from `mid+1` to `high`
}
`else`
{
  `return mid`
}



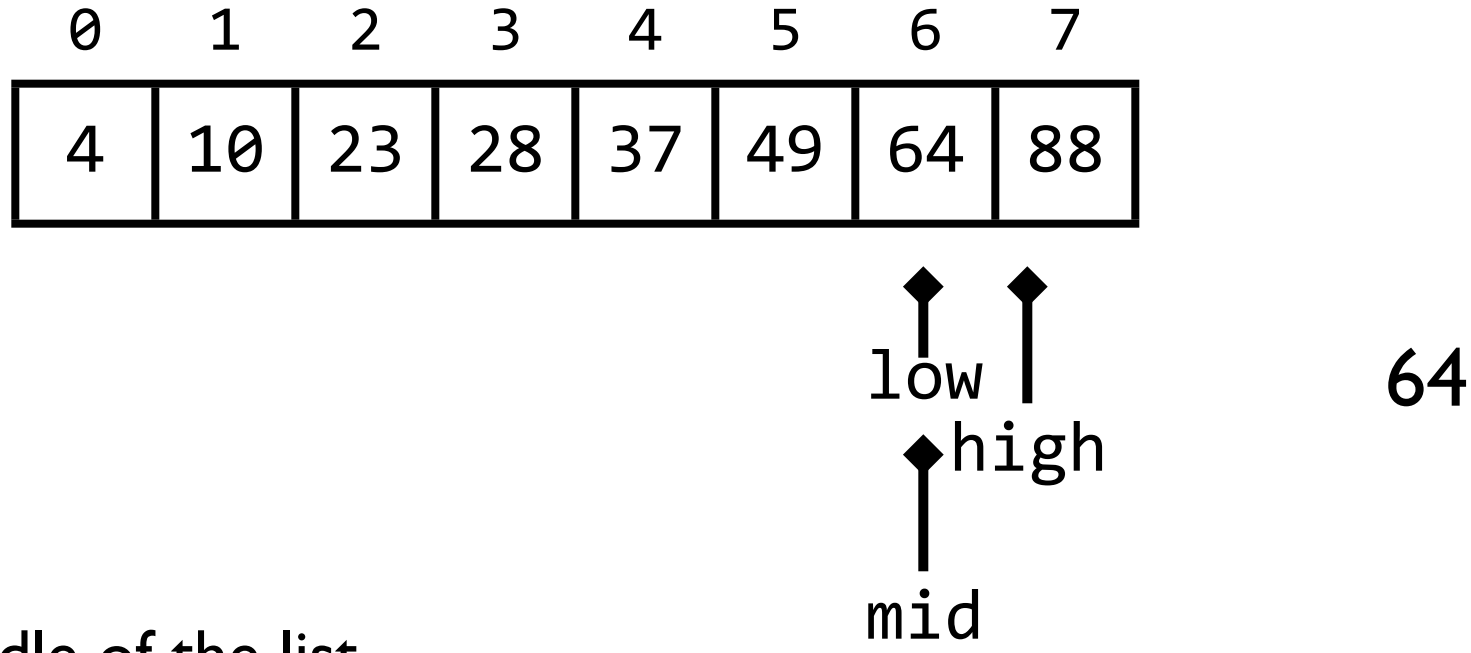
Find the middle of the list.
`if` (that item is larger than the item we're searching for)
{
  Binary search the list from `low` to `mid-1`
}
`else if` (that item is smaller than the item we're searching for)
{
  Binary search the list from `mid+1` to `high`
}
`else`
{
  `return mid`
}



Find the middle of the list.
`if` (that item is larger than the item we're searching for)
{
  Binary search the list from `low` to `mid-1`
}
`else if` (that item is smaller than the item we're searching for)
{
  Binary search the list from `mid+1` to `high`
}
`else`
{
  `return mid`
}

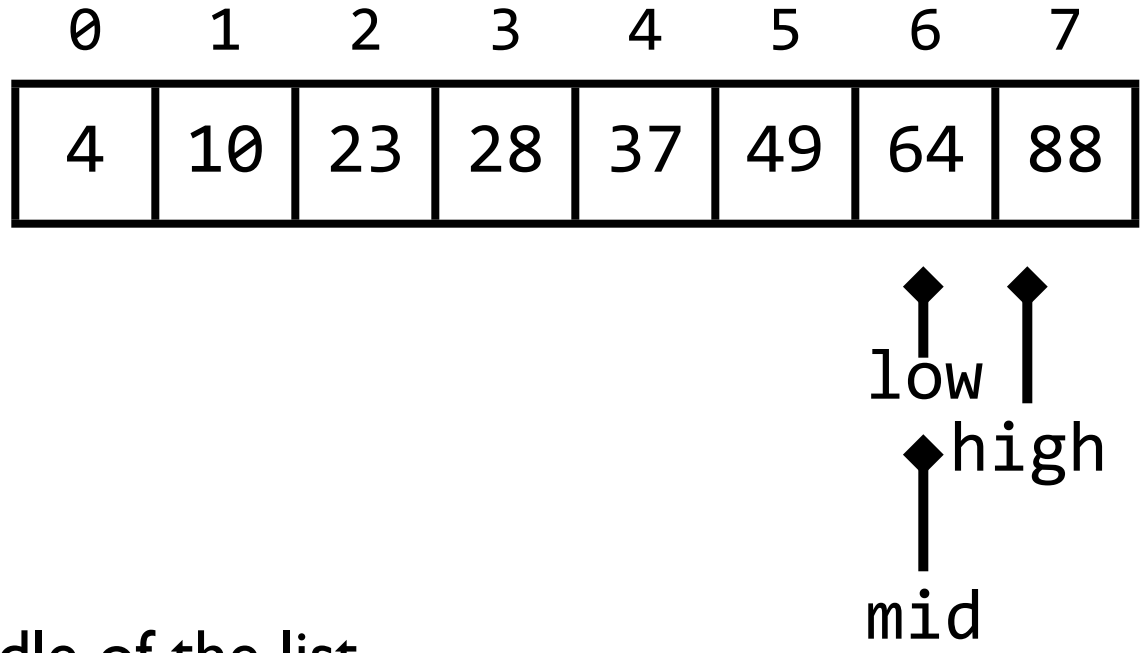


Find the middle of the list.
`if` (that item is larger than the item we're searching for)
{
  Binary search the list from `low` to `mid-1`
}
`else if` (that item is smaller than the item we're searching for)
{
  Binary search the list from `mid+1` to `high`
}
`else`
{
  `return mid`
}



Find the middle of the list.

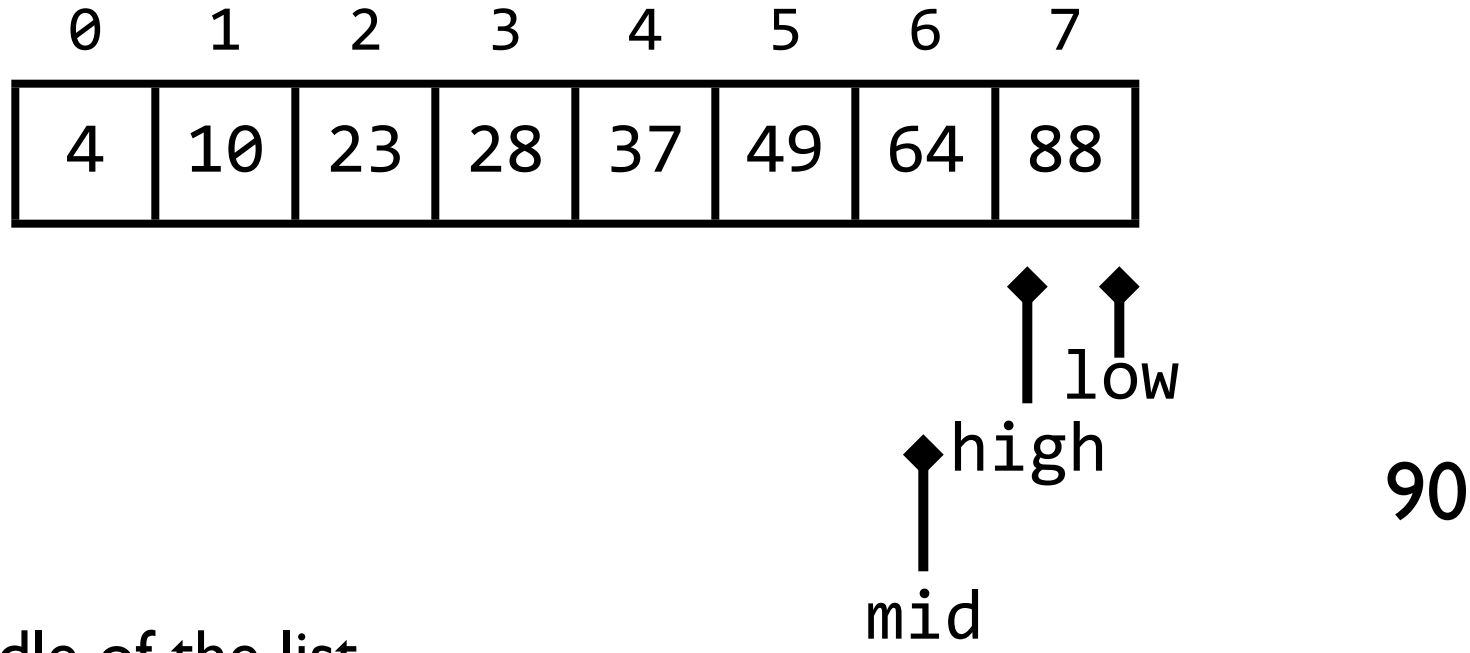
```
if (that item is larger than the item we're searching for)
{
   Binary search the list from low to mid-1
}
else if (that item is smaller than the item we're searching for)
{
   Binary search the list from mid+1 to high
}
else
{
   return mid
}
```



90

Find the middle of the list.

```
if (that item is larger than the item we're searching for)
{
   Binary search the list from low to mid-1
}
else if (that item is smaller than the item we're searching for)
{
   Binary search the list from mid+1 to high
}
else
{
   return mid
}
```



Find the middle of the list.

`if` (that item is larger than the item we're searching for)

{

  Binary search the list from `low` to `mid-1`

}

`else if` (that item is smaller than the item we're searching for)

{

  Binary search the list from `mid+1` to `high`

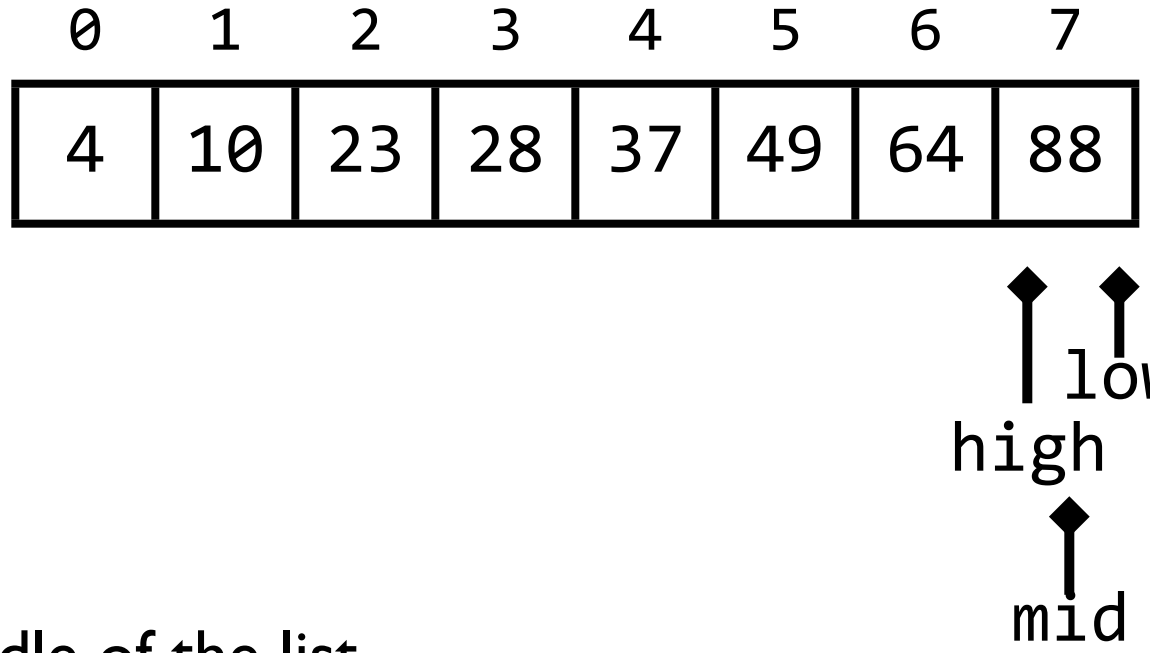
}

`else`

{

  `return mid`

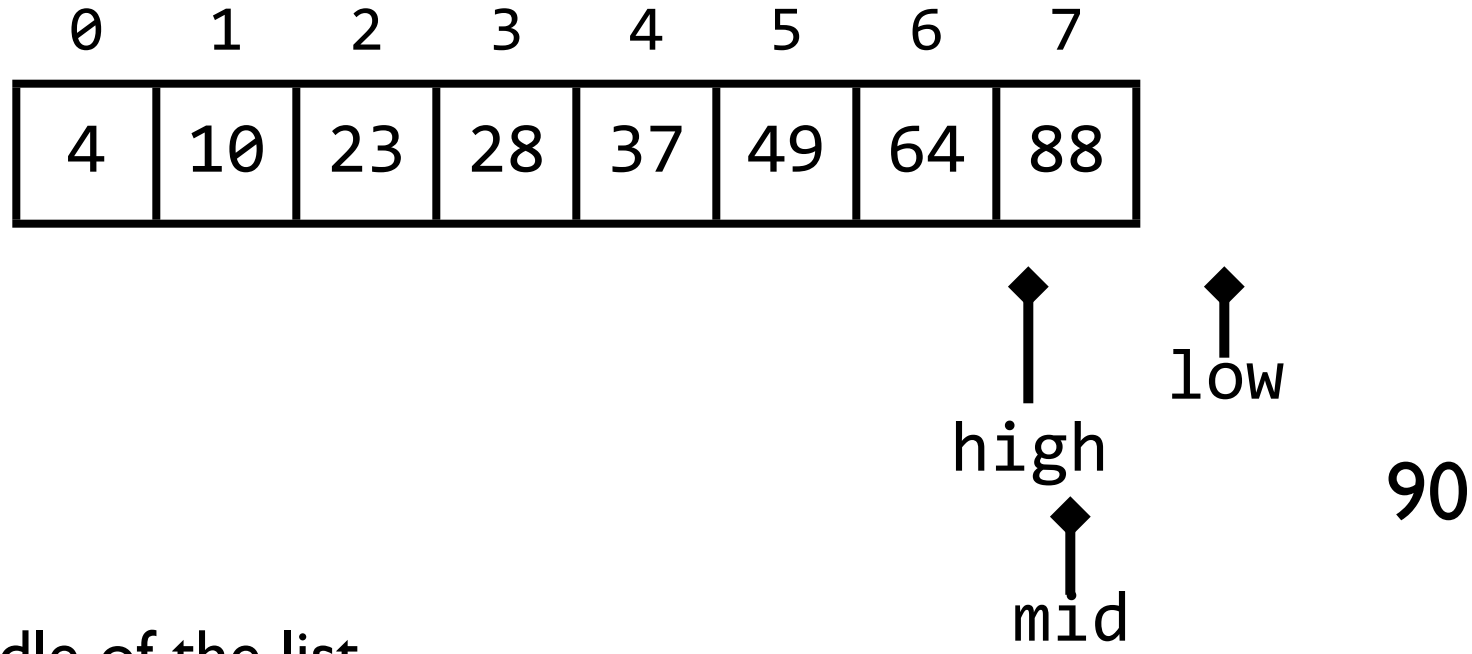
}



90

Find the middle of the list.

```
if (that item is larger than the item we're searching for)
{
   Binary search the list from low to mid-1
}
else if (that item is smaller than the item we're searching for)
{
   Binary search the list from mid+1 to high
}
else
{
   return mid
}
```



Find the middle of the list.

`if` (that item is larger than the item we're searching for)

```
{
```

Binary search the list from `low` to `mid-1`

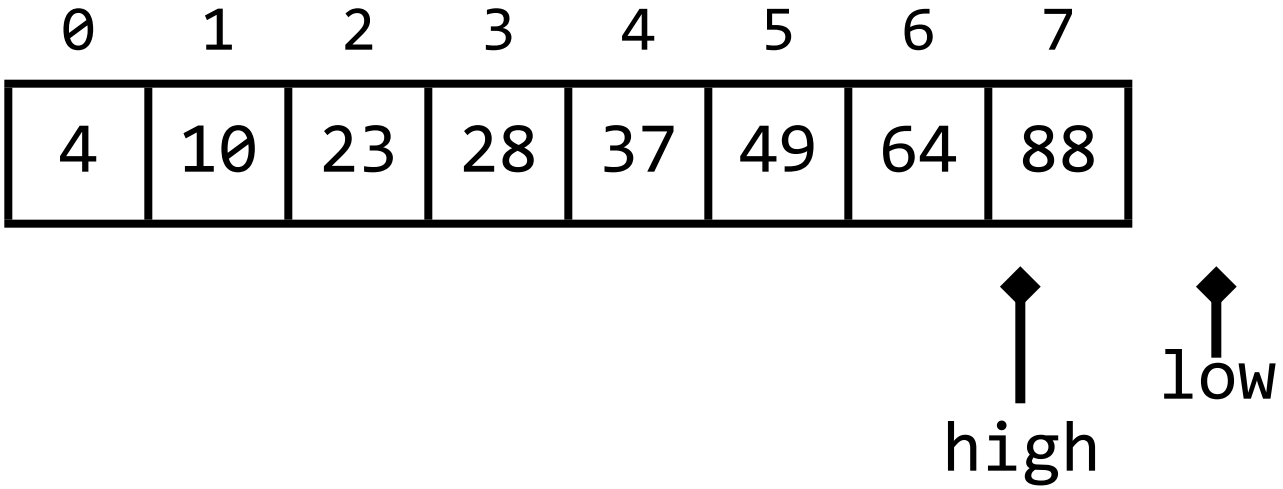
```
}
```

`else if` (that item is smaller than the item we're searching for)

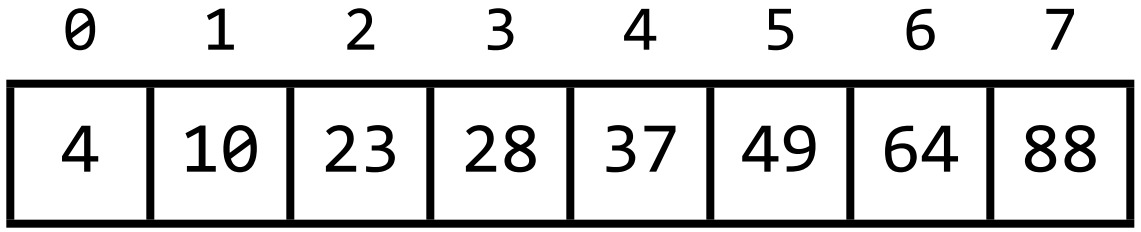
```
{
```

Binary search the list from `mid+1` to `high`

```
}
else
{
   return mid
}
```



Find the middle of the list.
`if` (that item is larger than the item we're searching for)
{
  Binary search the list from `low` to `mid-1`
}
`else if` (that item is smaller than the item we're searching for)
{
  Binary search the list from `mid+1` to `high`
}
`else`
{
  `return mid`
}



90

```
if (low > high)
{
   return -1
}
```

Find the middle of the list.

`if` (that item is larger than the item we're searching for)

```
{
```

Binary search the list from `low` to `mid-1`

```
}
```

`else if` (that item is smaller than the item we're searching for)

```
{
```

Binary search the list from `mid+1` to `high`

```
}
else
{
   return mid
}
```