APS105
Winter 2012

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

Lecture 32
April 4, 2012

Today

- Binary Search
- A5
- Sorting

- Course Evaluations on Monday
- CodeLab
- No lecture or tutorial on Friday

Searching

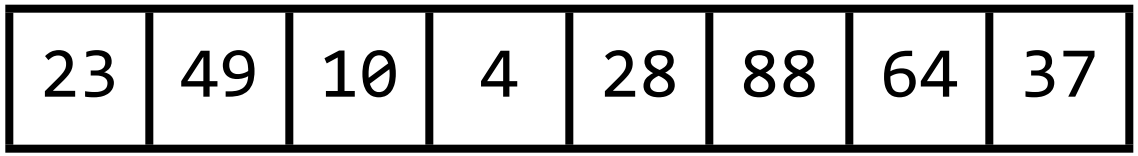
Searching

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

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 && index == -1; i++)
    {
        if (list[i] == x)
        {
            index = i;
        }
    }

    return index;
}
```

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

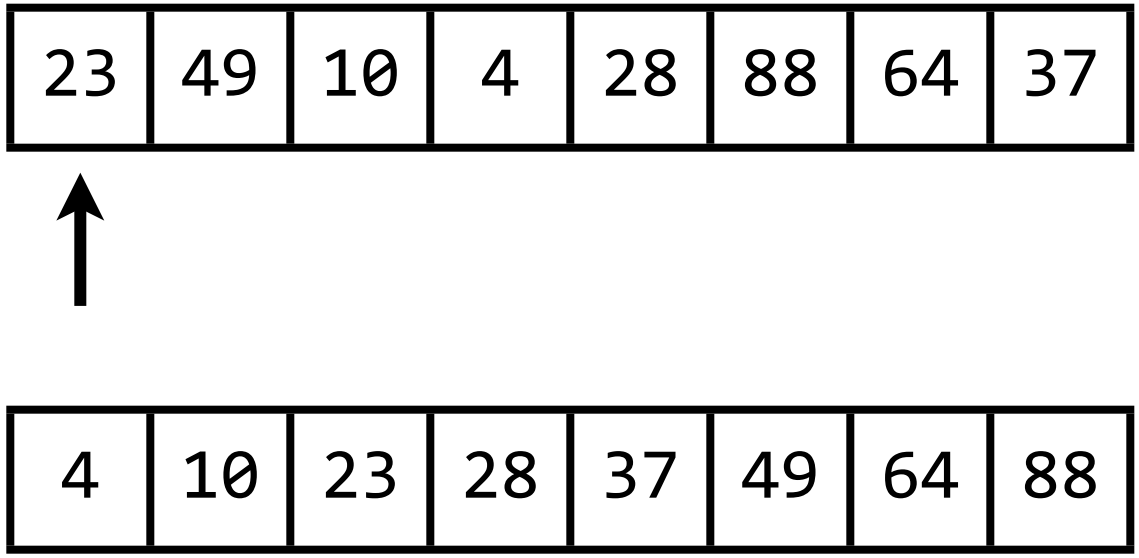
Sorted

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

23	49	10	4	28	88	64	37

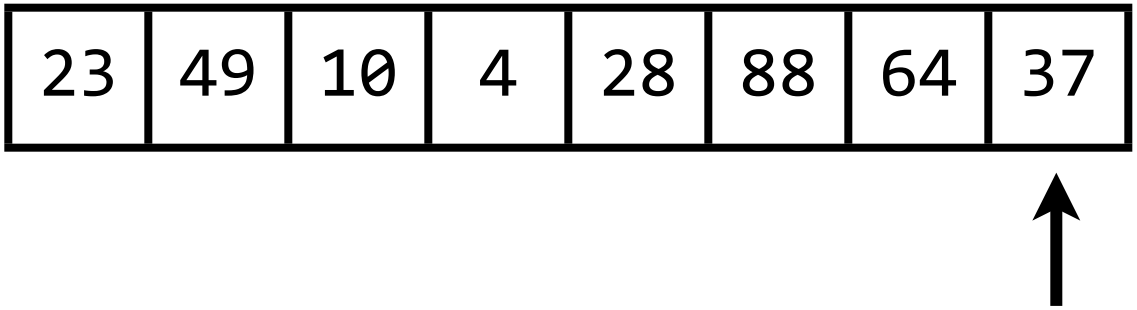
4	10	23	28	37	49	64	88

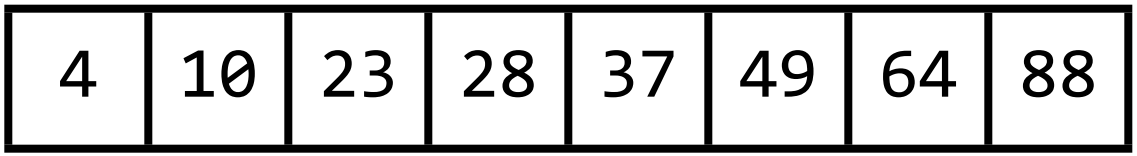
Unsorted vs. Sorted



Looking for 35

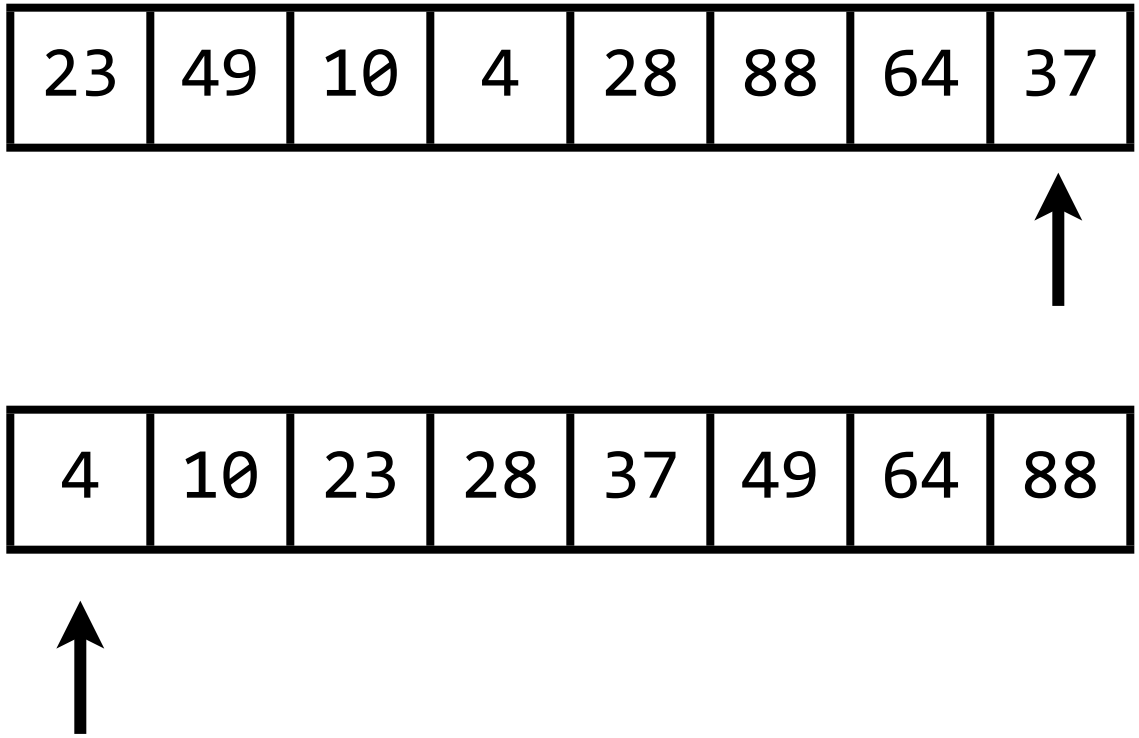
Unsorted vs. Sorted





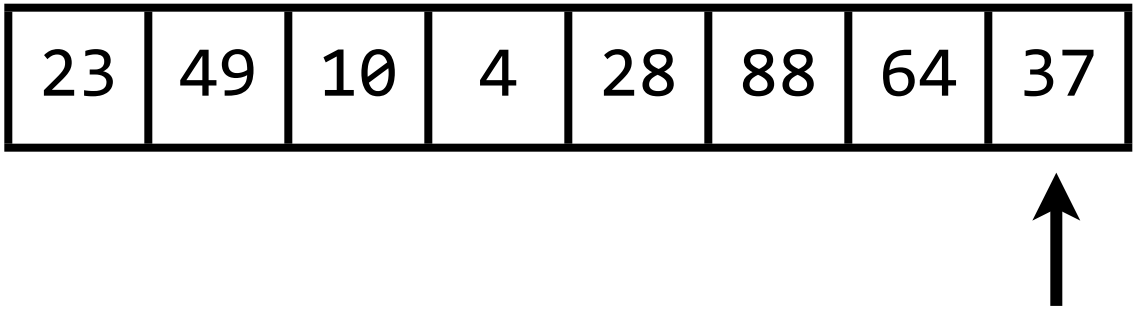
Looking for 35

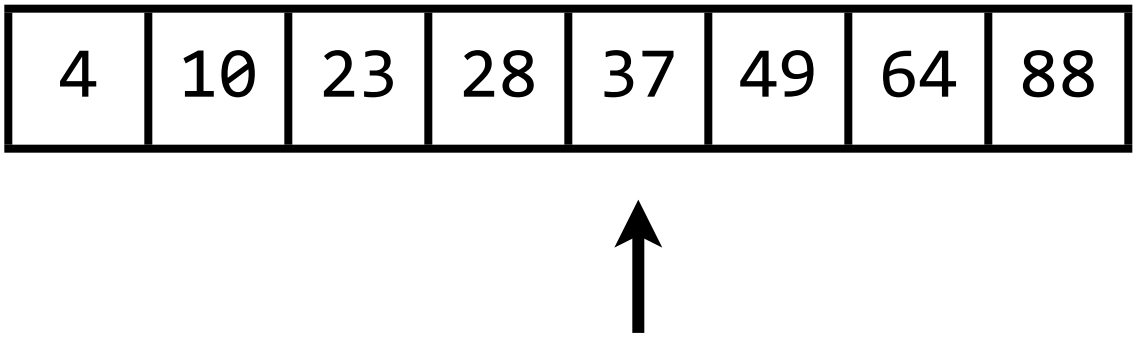
Unsorted vs. Sorted



Looking for 35

Unsorted vs. Sorted





Looking for 35

Sorted

- Might allow us to stop early

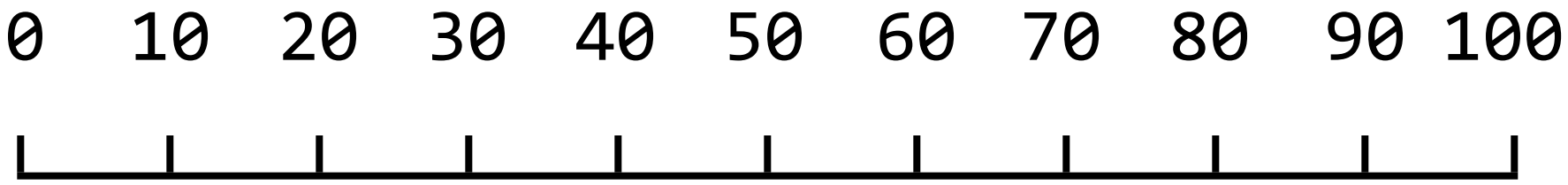
- Best case (same): it's the first item
- Worst case (same): it's the last item
- Variable case (new): it's not in the list
  - Might be able to stop after the first item, or might need to examine the entire list

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
                                  ——
```

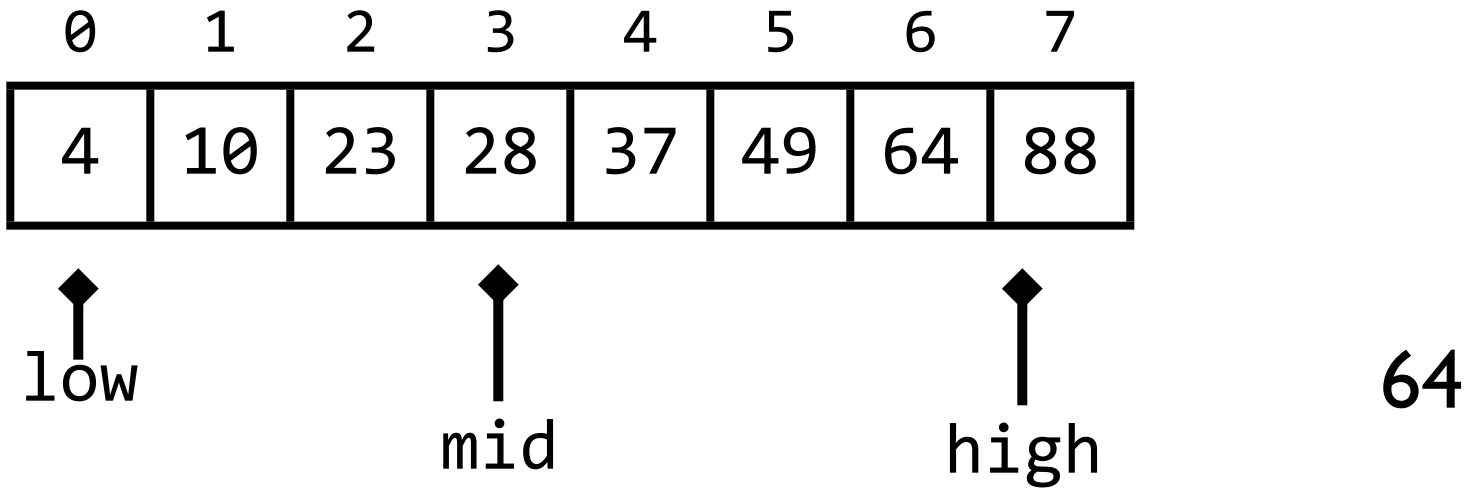
Guessing Game

- 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

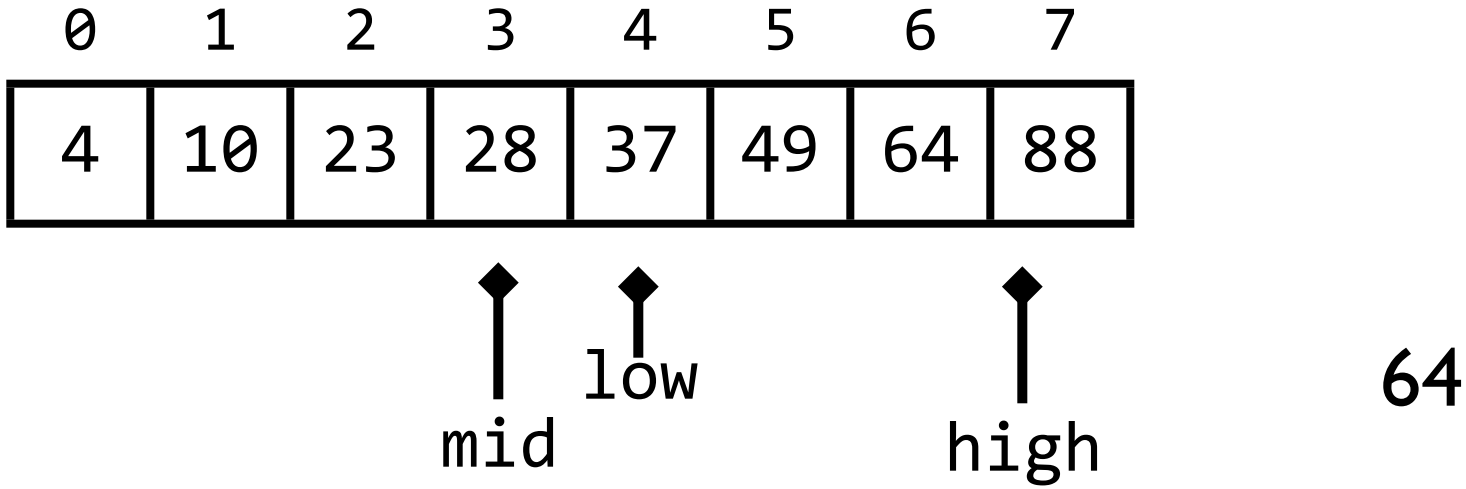
```
                                  50   75   87   81   84

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

Binary Search

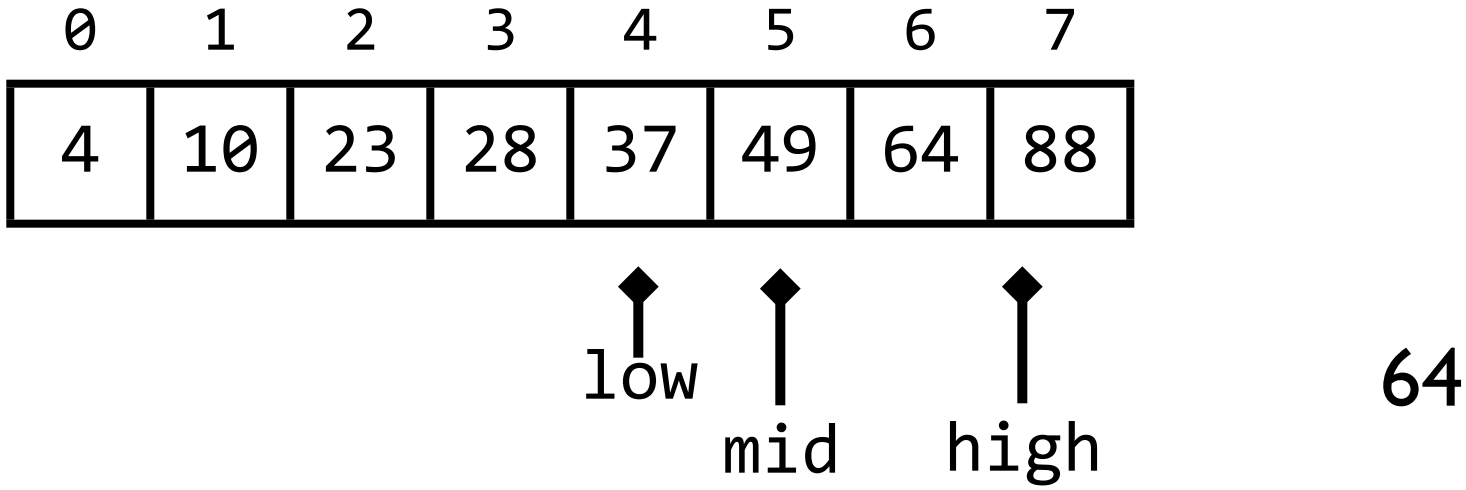
- This is an algorithm called binary search
- Each iteration cuts the search space in half
- Only works on sorted lists



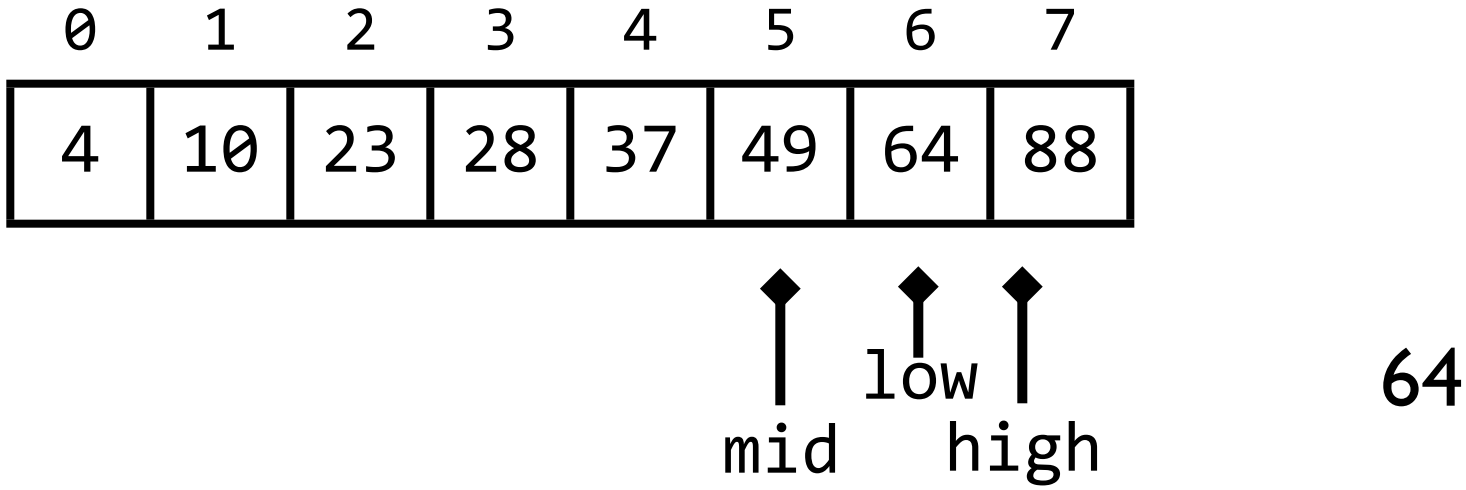
Find the middle of the list.
`if` (that item is larger than the item we're searching for)
{
  Binary search the list from `low` to `mid-1`
}
`else if` (that item is smaller than the item we're searching for)
{
  Binary search the list from `mid+1` to `high`
}
`else`
{
  `return mid`
}



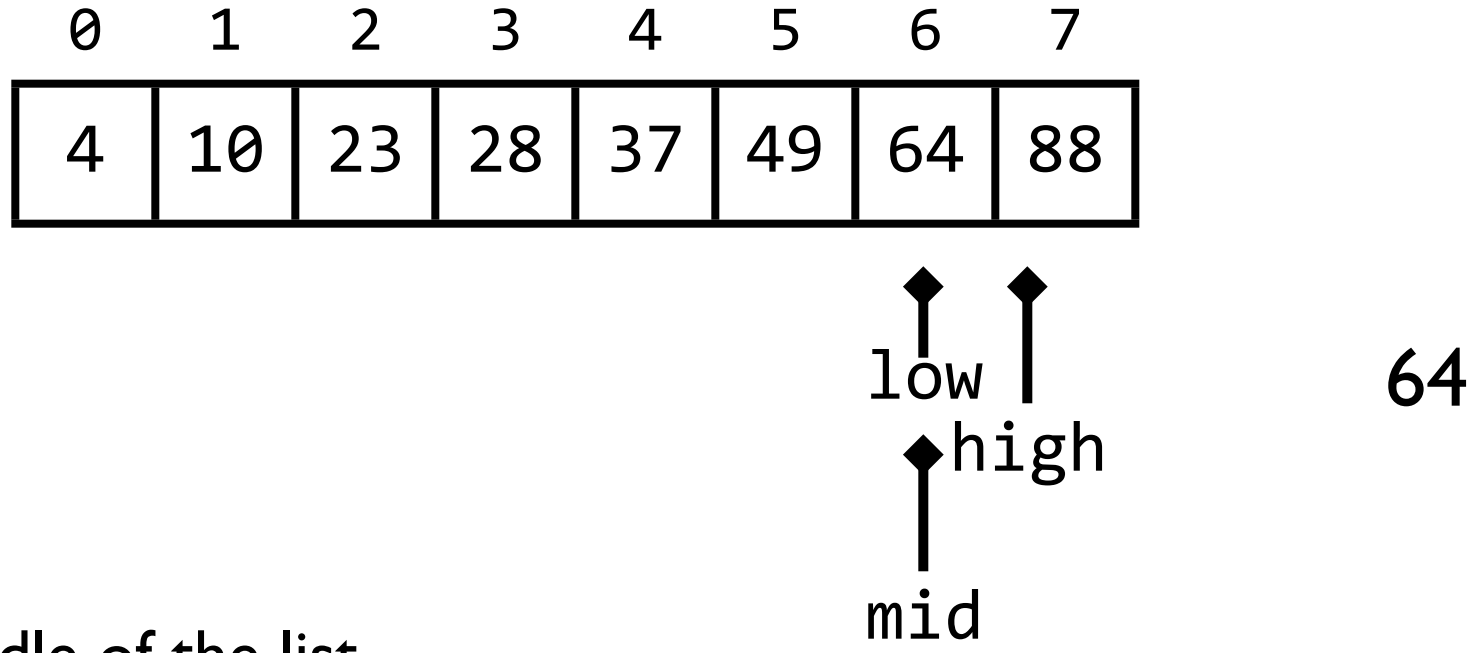
Find the middle of the list.
`if` (that item is larger than the item we're searching for)
{
  Binary search the list from `low` to `mid-1`
}
`else if` (that item is smaller than the item we're searching for)
{
  Binary search the list from `mid+1` to `high`
}
`else`
{
  `return mid`
}



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`
}

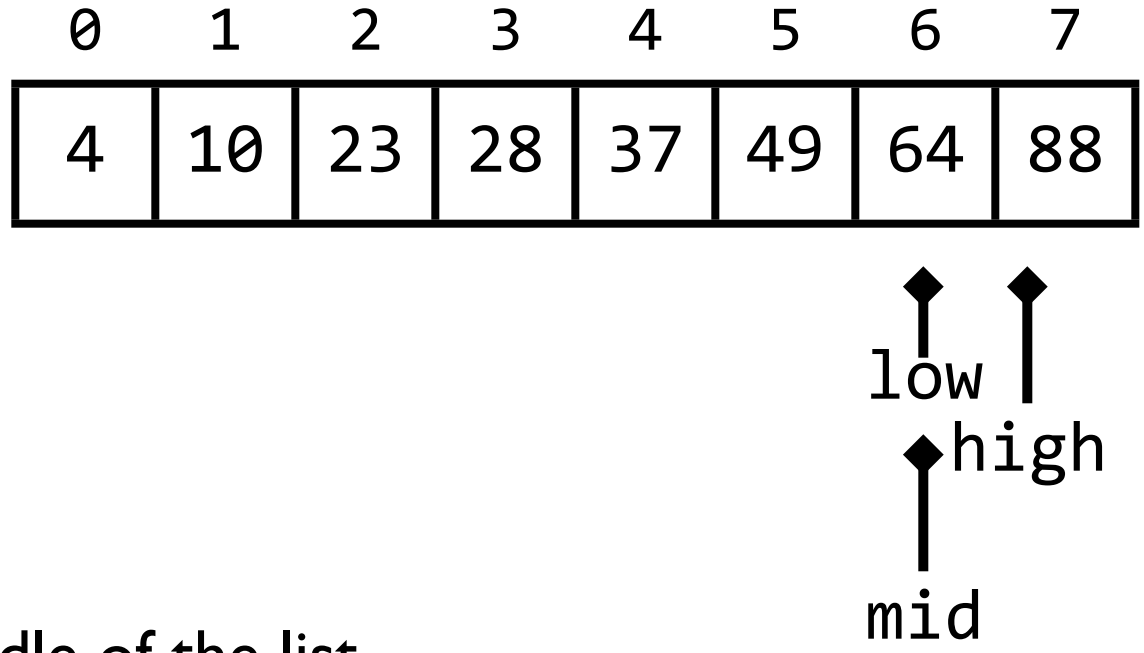


Find the middle of the list.
`if` (that item is larger than the item we're searching for)
{
  Binary search the list from `low` to `mid-1`
}
`else if` (that item is smaller than the item we're searching for)
{
  Binary search the list from `mid+1` to `high`
}
`else`
{
  `return mid`
}



Find the middle of the list.

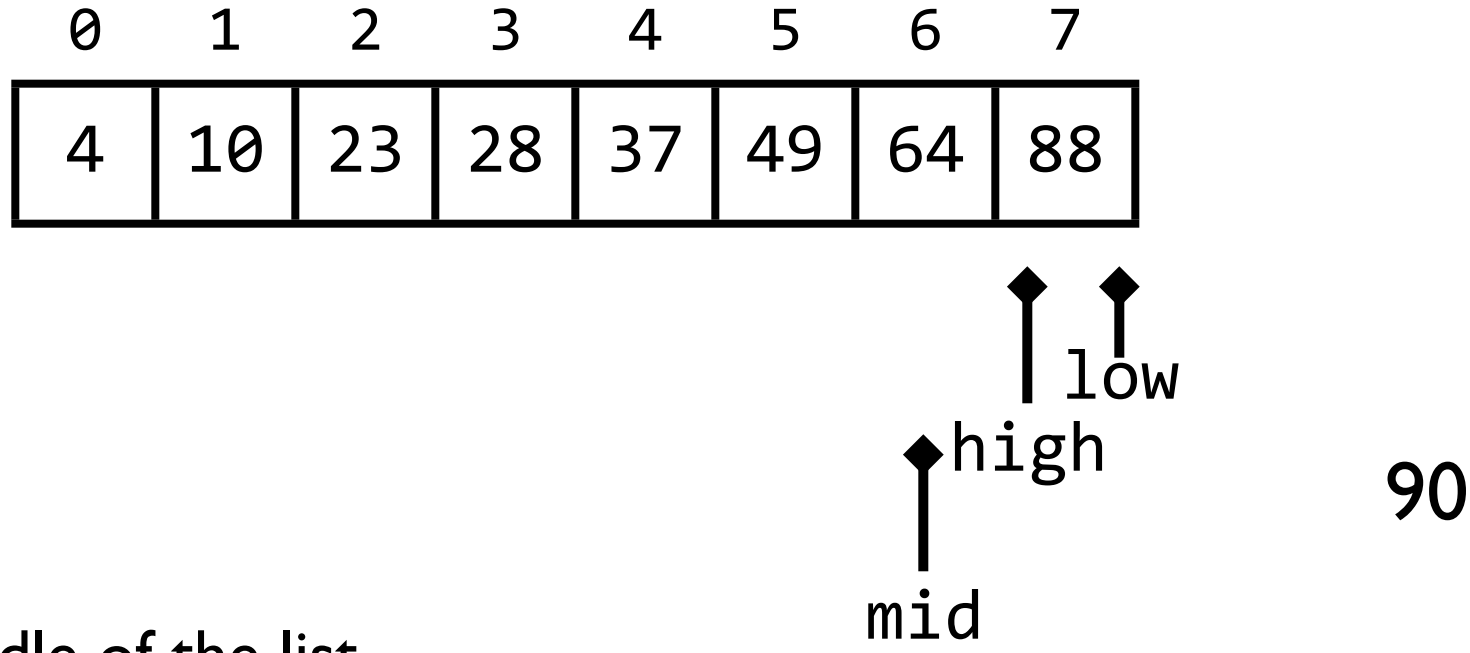
```
if (that item is larger than the item we're searching for)
{
   Binary search the list from low to mid-1
}
else if (that item is smaller than the item we're searching for)
{
   Binary search the list from mid+1 to high
}
else
{
   return mid
}
```



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.

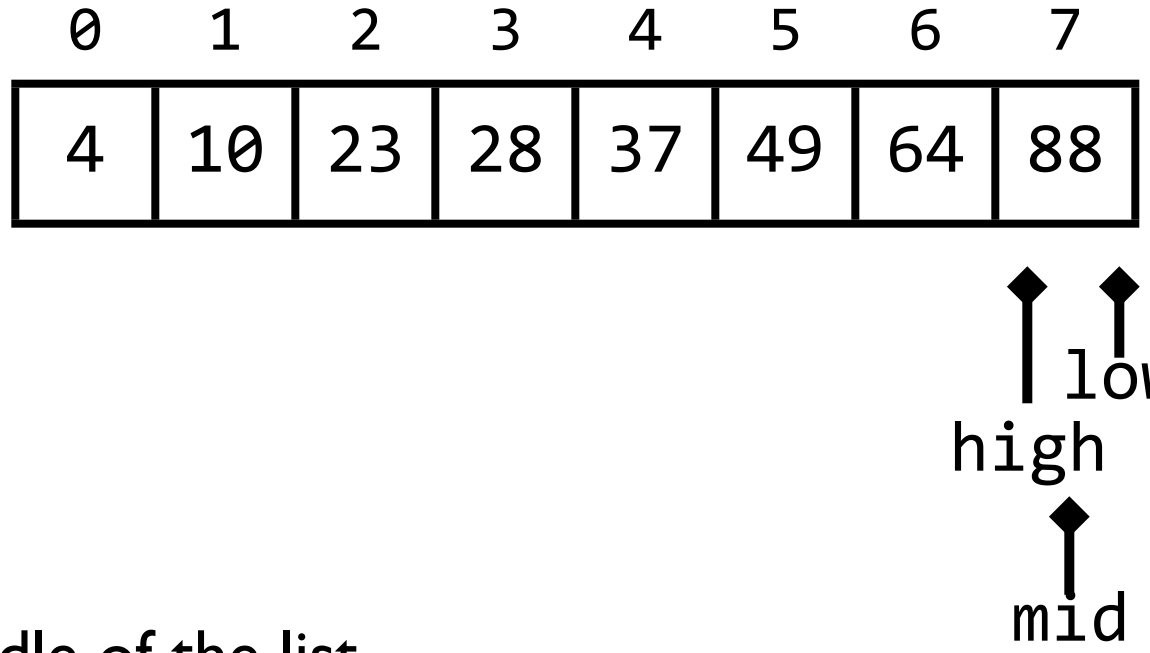
`if` (that item is larger than the item we're searching for)

```
{
   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
}
```

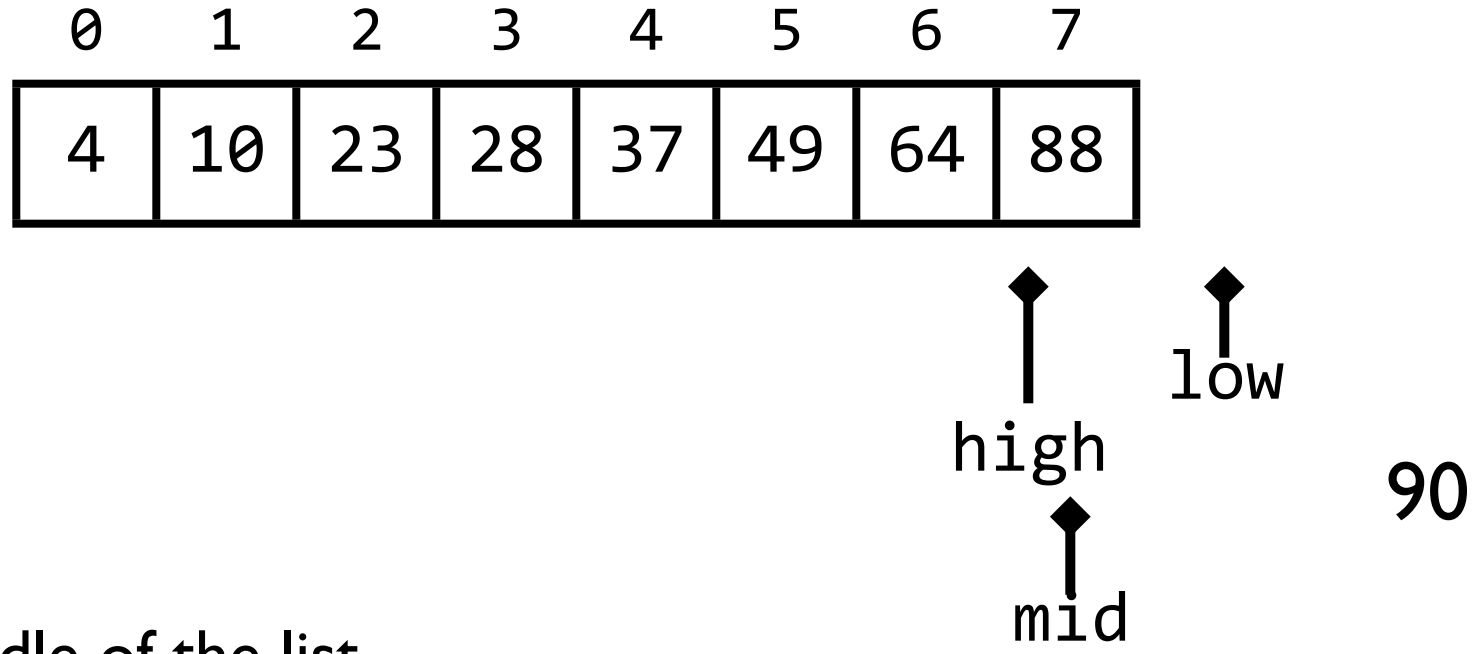
```
else
{
   return mid
}
```



90

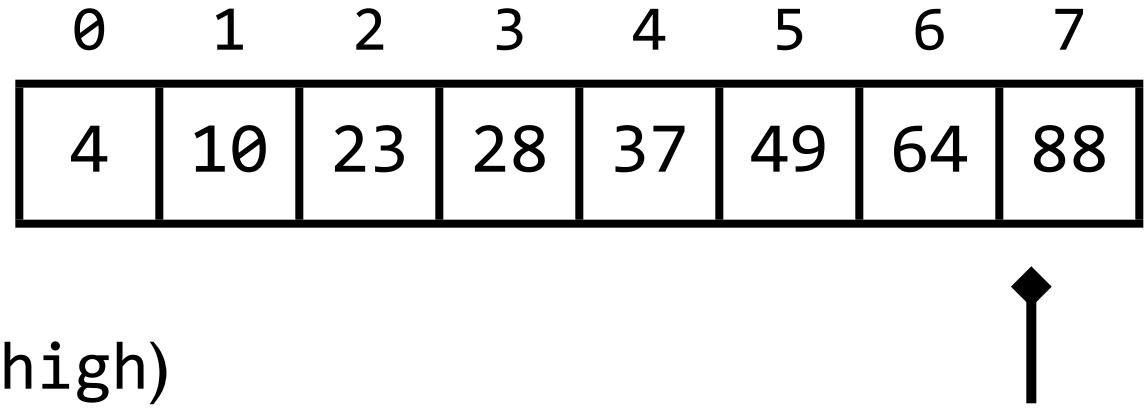
Find the middle of the list.

```
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
}
```



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
}
```

```
int binSearch(int list[], int n, int low, int high, int x)
{
   if (low > high)
   {
      return -1;
   }

   int mid = (low + high) / 2;

   if (list[mid] > x)
   {
      return binSearch(list, n, low, mid - 1, x);
   }
   else if (list[mid] < x)
   {
      return binSearch(list, n, mid + 1, high, x);
   }
   else
   {
      return mid;
   }
}
```

Linked Lists

- Can we use binary search on a linked list?
- No, since we need to access `i`th element

Binary Search Performance

- Each search cuts the list in half

$$N \quad \frac{N}{2} \quad \frac{N}{4} \quad \frac{N}{8} \quad \ldots \quad \frac{N}{2^t}$$

$$\frac{N}{2^t} = 1$$

$$2^t = N$$

$$t = \frac{\log N}{\log 2}$$

$$t = \log_2 N$$

Binary vs. Sequential Search

- Sequential search increases linearly with N
- Binary search increases with $\log_2(N)$

N	*Binary*	*Sequential*
10	3	10
100	6	100
1,000	10	1,000
10,000	13	10,000
100,000	16	100,000
100,000,000	26	100,000,000
100,000,000,000	36	100,000,000,000

~ 30 ns ~ 100 seconds

Assignment 5

Assignment 5

- Largest sum of subset(s)

`{1, 2, 5, 6}` Target value of 6

Largest sum of a subset that is
less than or equal to 6 is 6

Two subsets have a sum of 6

`{1, 5}` and `{6}`

Structure

- Get a set of numbers from the user (or use default)
- Get a target value
- (Recursively) generate a list of all possible subsets
- Go through that list to find the sum of a subset that is closest to the target value
- Go through that list again and print out all subsets that have that sum

Permutations

- Need a data structure to deal with a "list" of subsets
- First part of assignment gets you to build some linked list tools
- Once you have those, you move on to actually generating the permutations
- `debugPrintList()` is for helping you debug, not for printing the final list of subsets
- You will want to add some more list helper functions (which must also be recursive)

Permutations

Create an empty results list.
`if` (the length of `numbers` is `1`)
{
  Add a `Node` containing that single number to the results list.
}
`else`
{
  Take the first number off of `numbers`.
  Recursively generate the permutations of the remaining array.
  Recursively traverse that list, and create a new list consisting of subsets
    obtained by adding the removed number to every subset in the list.
  Add that list to our results list.
  Add the (unmodified) list of recursively generated subsets to results list.
  Add a subset consisting of the single number we removed to results list.
}
Return the results list.

```
generatePermutations({1, 2, 3})
```

(This is pseudocode, not valid C!)

```
generatePermutations({1, 2, 3})
```

```
generatePermutations({1, 2, 3})
   1
```

```
generatePermutations({1, 2, 3})
  1


 generatePermutations({2, 3})
```

```
generatePermutations({1, 2, 3})
   1


 generatePermutations({2, 3})
    2
```

```
generatePermutations({1, 2, 3})
  1


 generatePermutations({2, 3})
    2


   generatePermutations({3})
```

```
generatePermutations({1, 2, 3})
  1

 generatePermutations({2, 3})
    2

   generatePermutations({3})
                          {3}
```

```
generatePermutations({1, 2, 3})
   1


 generatePermutations({2, 3})
    2
              {3}
```

```
generatePermutations({1, 2, 3})
  1

 generatePermutations({2, 3})
    2                          {3, 2}
             {3}
```

```
generatePermutations({1, 2, 3})
  1

 generatePermutations({2, 3})
    2                              {3, 2} {3}
             {3}
```

```
generatePermutations({1, 2, 3})
  1

 generatePermutations({2, 3})
    2                              {3, 2} {3} {2}
             {3}
```

```
generatePermutations({1, 2, 3})
   1

       {3, 2} {3} {2}
```

```
generatePermutations({1, 2, 3})
   1
                              {3, 2, 1}
      {3, 2} {3} {2}
```

```
generatePermutations({1, 2, 3})
   1
                              {3, 2, 1}  {3, 1}
      {3, 2} {3} {2}
```

```
generatePermutations({1, 2, 3})
   1
                              {3, 2, 1}  {3, 1}  {2, 1}
      {3, 2} {3} {2}
```

```
generatePermutations({1, 2, 3})
   1
                              {3, 2, 1}  {3, 1}  {2, 1}
      {3, 2} {3} {2}            {3, 2}
```

```
generatePermutations({1, 2, 3})
   1
                              {3, 2, 1}  {3, 1}  {2, 1}
      {3, 2} {3} {2}            {3, 2}     {3}
```

```
generatePermutations({1, 2, 3})
   1
                                {3, 2, 1}  {3, 1}  {2, 1}
      {3, 2} {3} {2}              {3, 2}     {3}     {2}
```

```
generatePermutations({1, 2, 3})
   1
                                {3, 2, 1}  {3, 1}  {2, 1}
      {3, 2} {3} {2}              {3, 2}     {3}     {2}
                                    {1}
```

{1, 2, 3}

Target value of 3

{3, 2, 1}
{3, 2}
{1}

{3, 1}
{3}

{2, 1}
{2}

{1, 2, 3}

Target value of 3

{3, 2, 1}
{3, 2}
{1}

{3, 1}
{3}

{2, 1}
{2}

Largest sum <= 3: 3

{1, 2, 3}

Target value of 3

{3, 2, 1} {3, 1} {2, 1}

{3, 2} {3} {2}

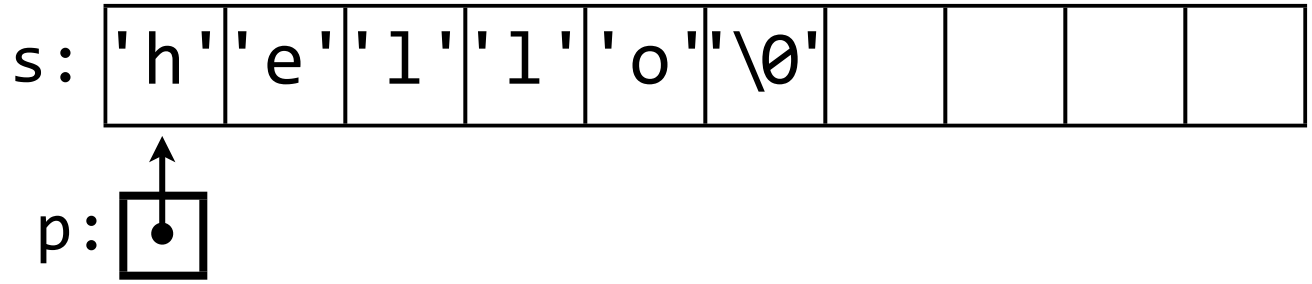
{1}

Largest sum <= 3: 3

{2, 1} {3}

Dealing with Sub-Arrays

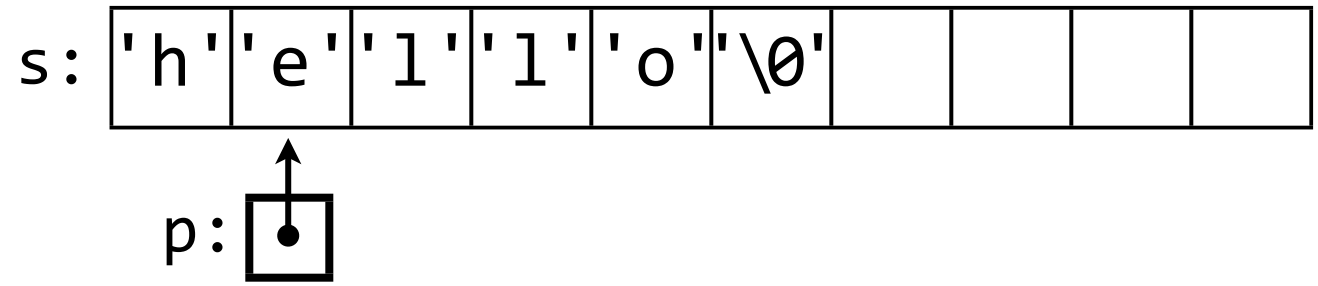
```
char s[9 + 1] = "hello";
```




```
char *p = s;      char *p = &s[0];
```

Dealing with Sub-Arrays

```
char s[9 + 1] = "hello";
```

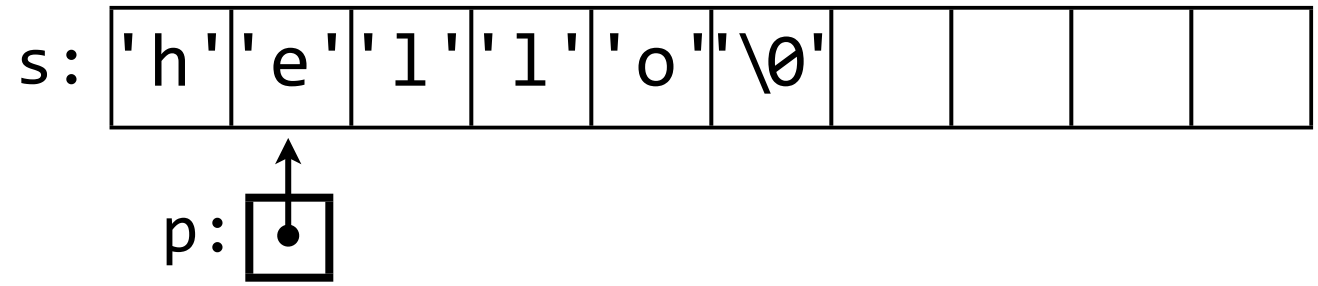



```
char *p = s;
p = &s[1];
```

```
char *p = &s[0];
p = s + 1;
```

Dealing with Sub-Arrays

```
char s[9 + 1] = "hello";
```




```
char *p = s;        char *p = &s[0];
p = &s[1];          p = s + 1;
printf("%s",s);     hello
printf("%s",p);     ello
strlen(s);          5
strlen(p);          4
strlen(&s[1]);      4
strlen(s[1]);
```

Wrong

Sorting

Sorting

- Two approaches:
  - Keep your list sorted as you add items
  - Take an unsorted list and sort it

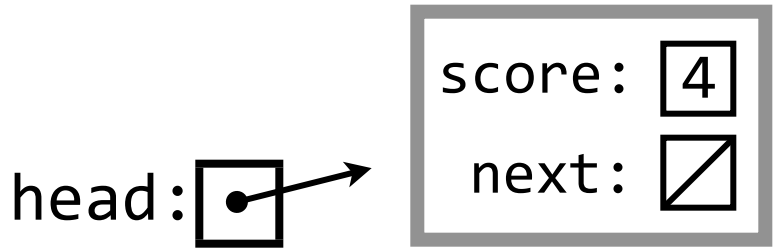
- Depends on how often you're inserting items

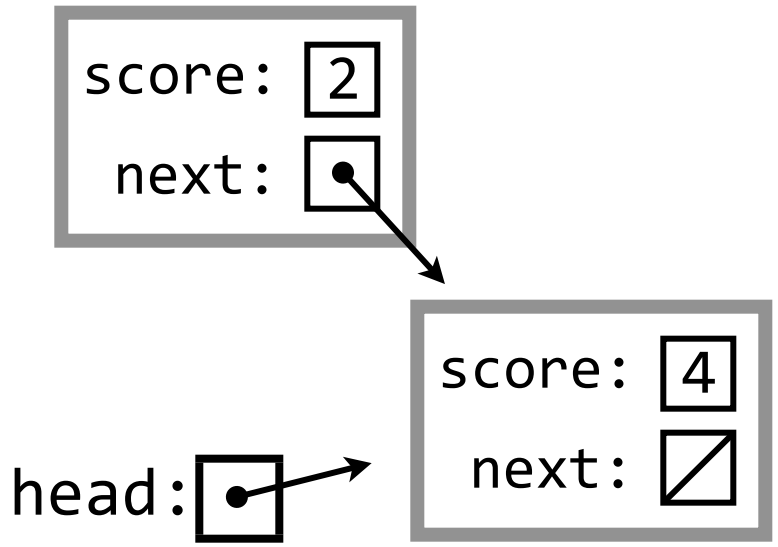
Keep It Sorted

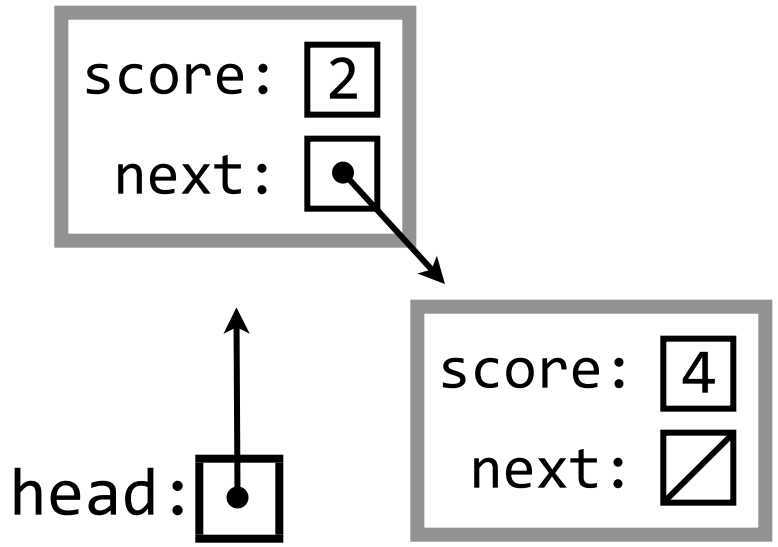
- Most commonly used with linked lists

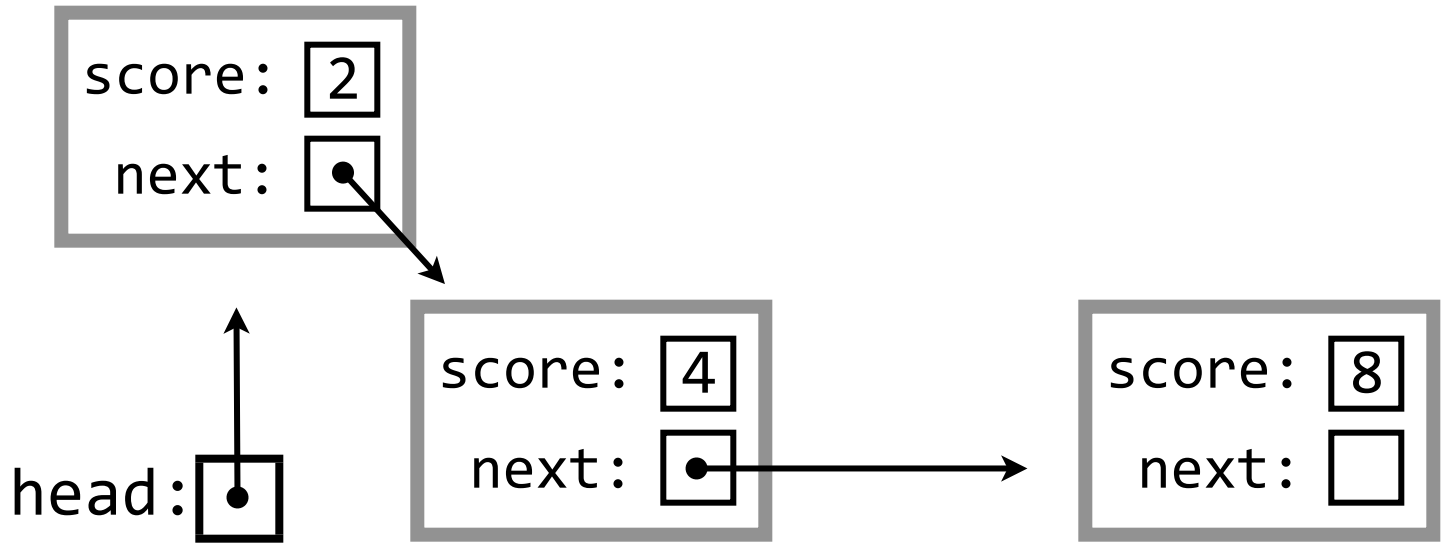
Insert in order:

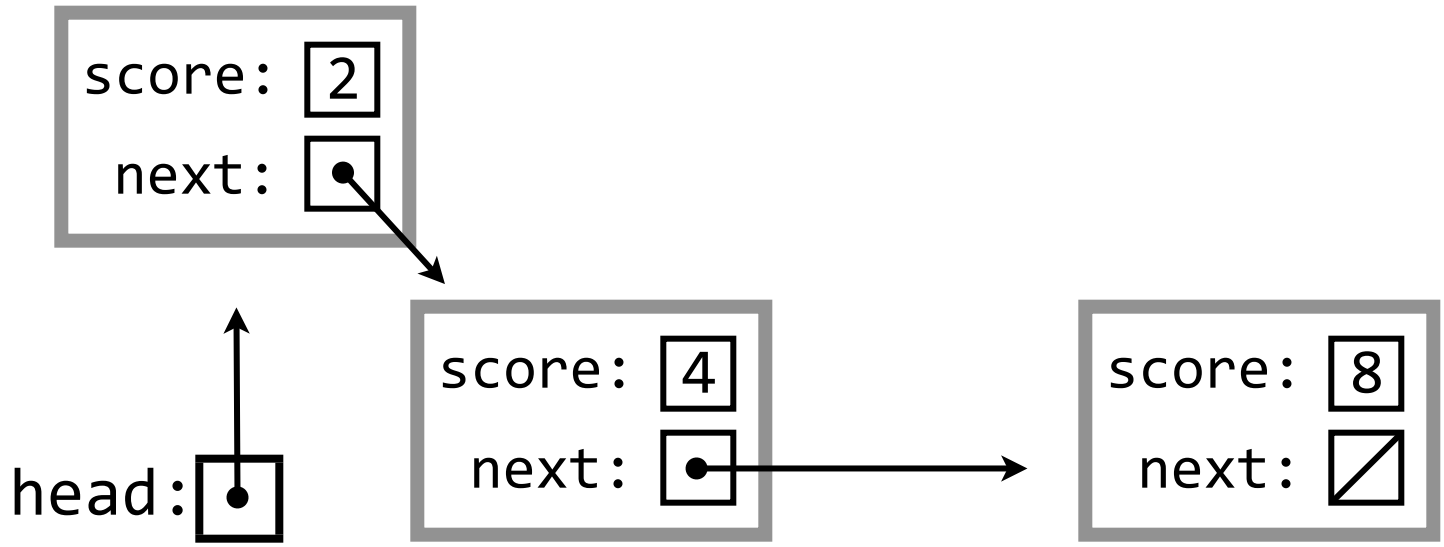
Find the node that should come before the new node.
Set the new node's pointer to that node's next pointer.
Set that node's pointer to the new node.

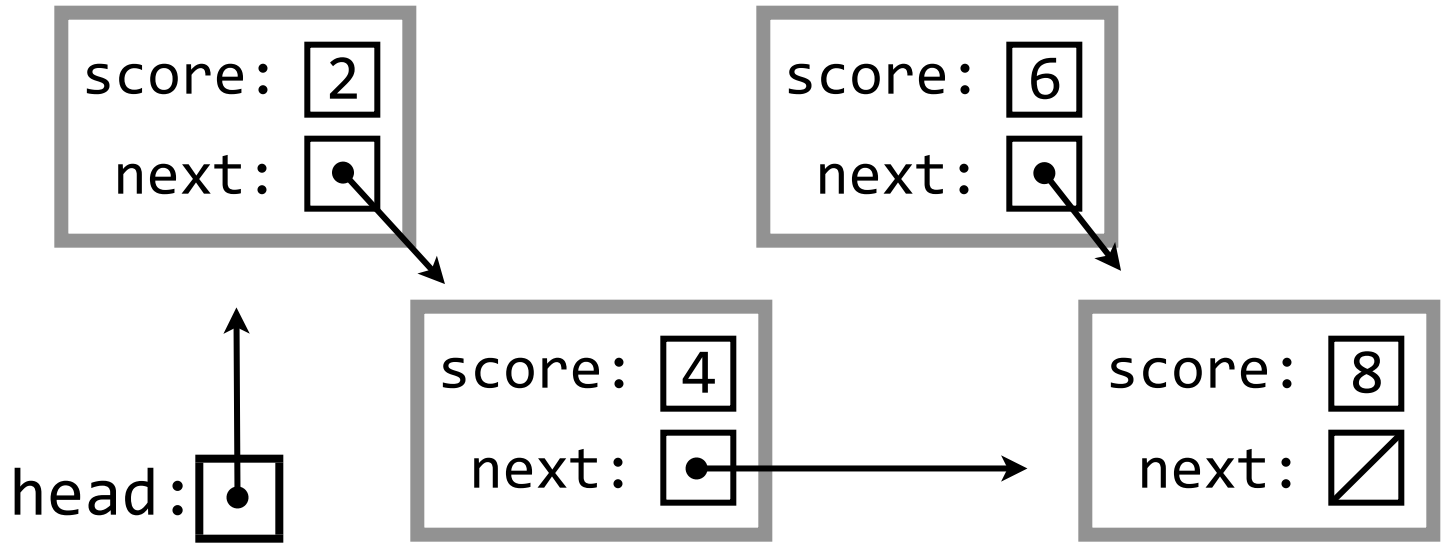

score: 4
next:
head:

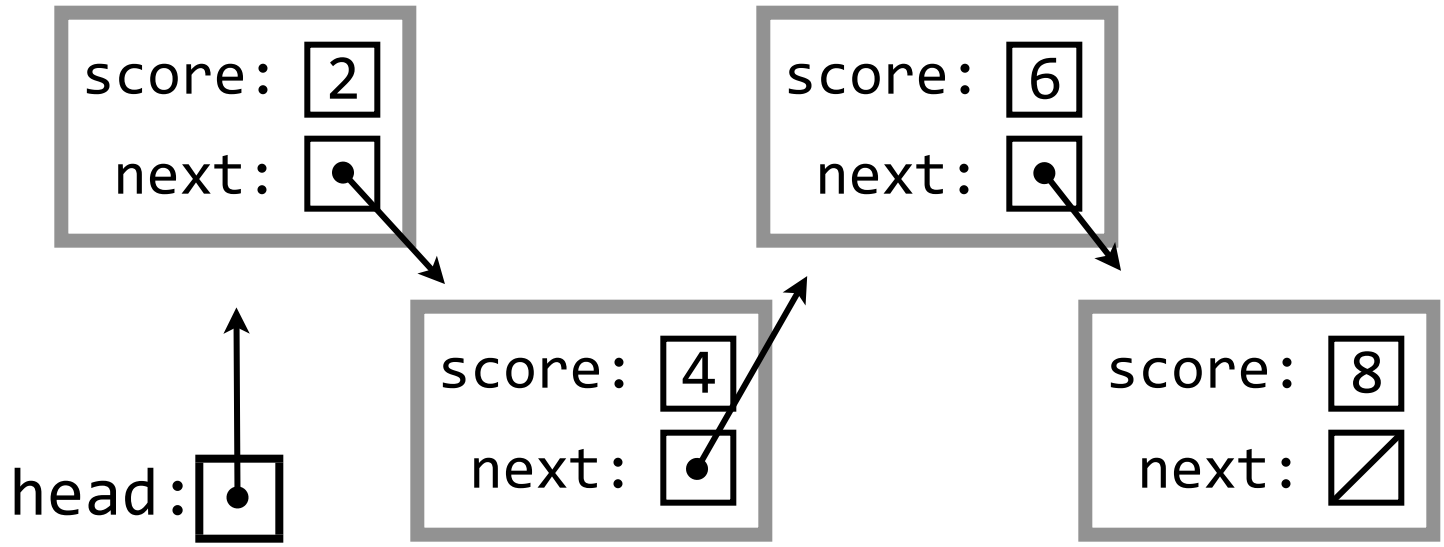

score: 2
next:
score: 4
next:
head:


score: 2
next:
score: 4
next:
head:


score: 2
next:
score: 4
next:
score: 8
next:
head:


score: 2
next:
score: 4
next:
score: 8
next:
head:


score: 2
next:
score: 6
next:
score: 4
next:
score: 8
next:
head:


score: 2
next:
score: 4
next:
score: 6
next:
score: 8
next:
head:

```
Node *insertSorted(Node *head, Node *newNode)
{
  Node *current = head;
  Node *prev = NULL;

  while (current != NULL && current->score < newNode->score)
  {
     prev = current;
     current = current->next;
  }
  if (head == NULL || prev == NULL)
  {
     newNode->next = head;
     return newNode;
  }
  else
  {
     newNode->next = current;
     prev->next = newNode;
     return head;
  }
}
```

Sorting an Existing List

- Simpler but inefficient algorithms
  - e.g., bubble sort, selection sort, insertion sort
- Complicated but faster algorithms
  - e.g., merge sort, quicksort
- In the real world, you rarely write these yourself

Bubble Sort

- Simplest of the "exchange sorts"

```
while (the list is unsorted)
{
    Go through the list, and compare pairs of items.
    if (they're out of order)
    {
        Swap them.
    }
}
```

23 49 10 4 28 88 64 37

23 49 10 4 28 88 64 37

23 10 49 4 28 88 64 37

23 10 49 4 28 88 64 37

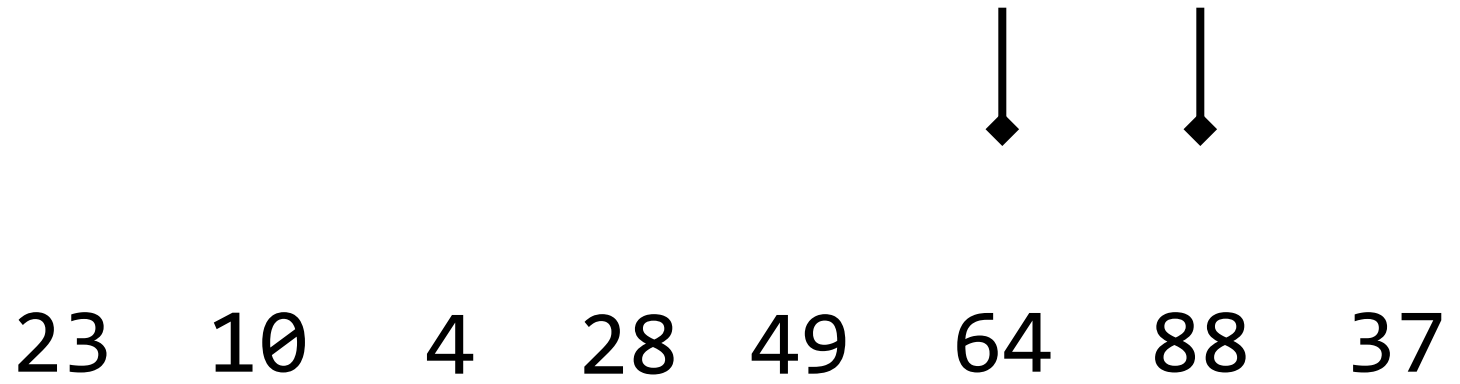
23 10 4 49 28 88 64 37

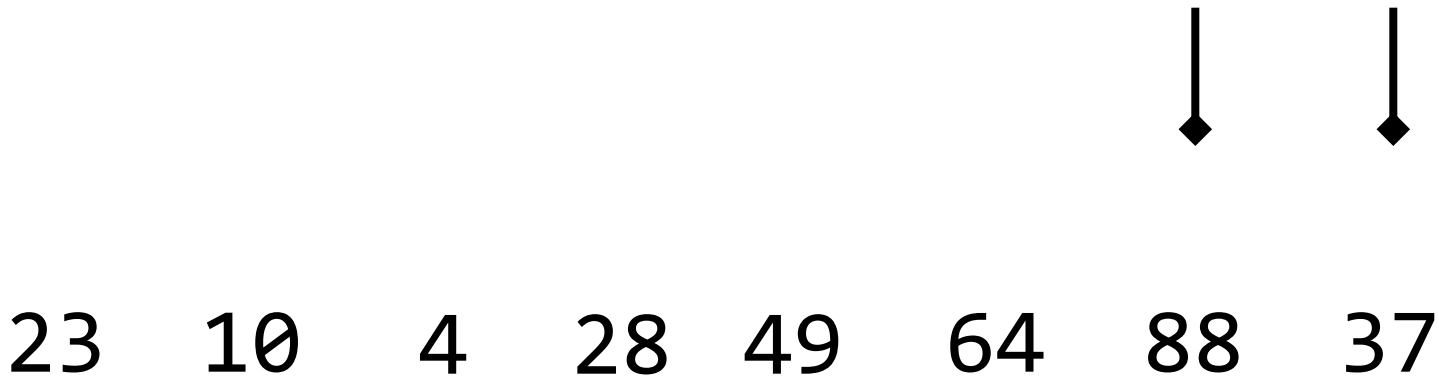
23 10 4 49 28 88 64 37

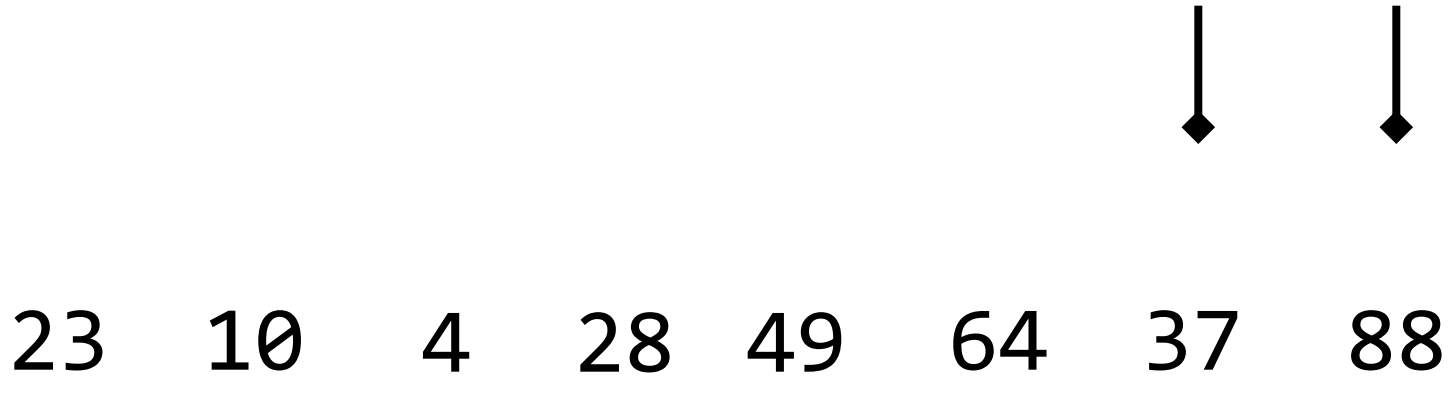
23 10 4 28 49 88 64 37

23 10 4 28 49 88 64 37

23 10 4 28 49 88 64 37


23 10 4 28 49 64 88 37


23 10 4 28 49 64 88 37


23 10 4 28 49 64 37 88

23 10 4 28 49 64 37 88

23 10 4 28 49 64 37 88

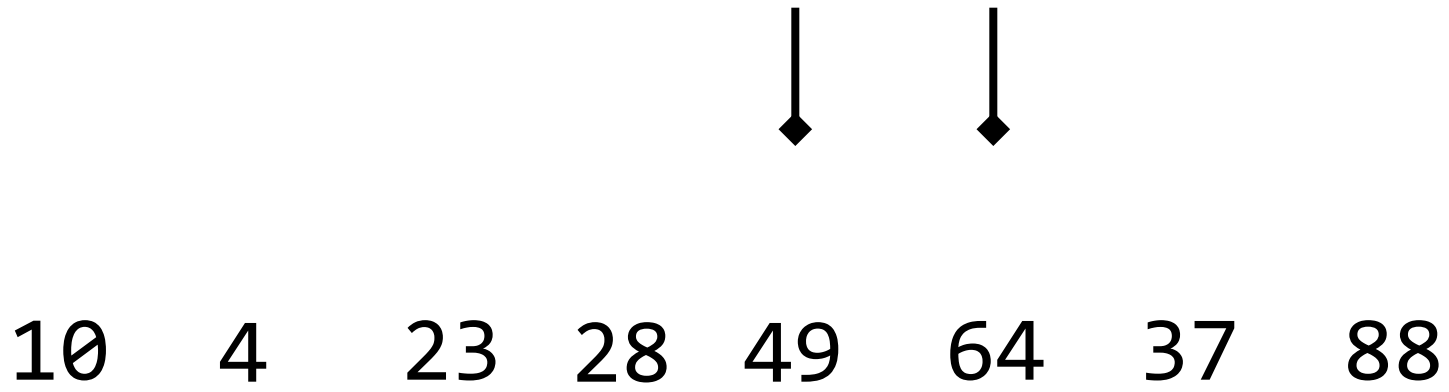
10 23 4 28 49 64 37 88

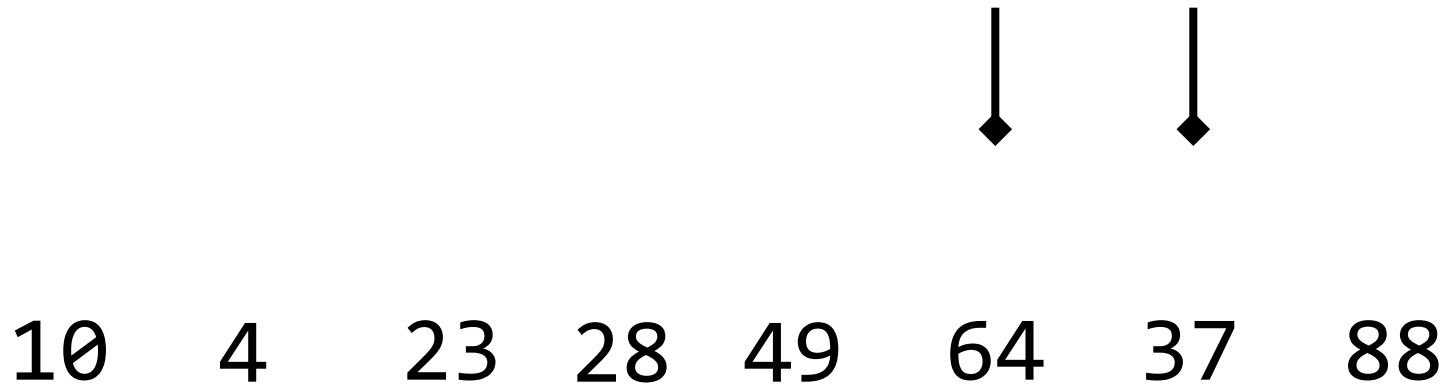
10 23 4 28 49 64 37 88

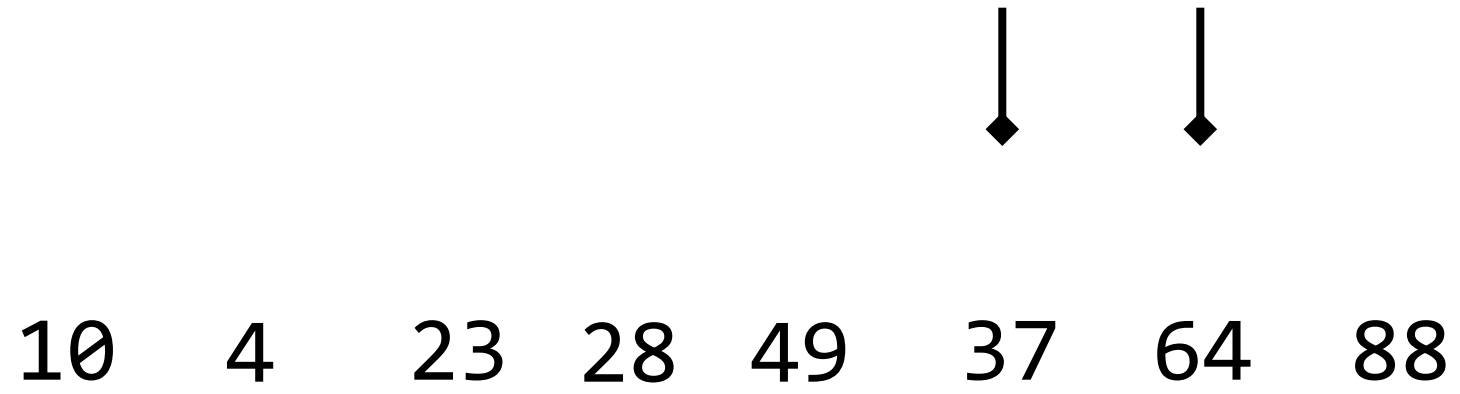
10 4 23 28 49 64 37 88

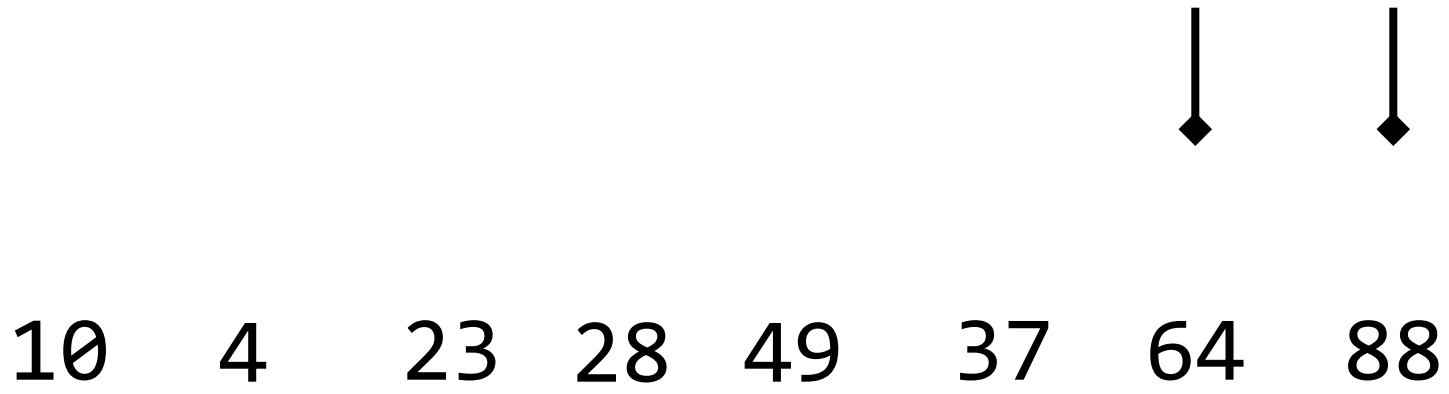
10 4 23 28 49 64 37 88

10 4 23 28 49 64 37 88


10 4 23 28 49 64 37 88


10 4 23 28 49 64 37 88


10 4 23 28 49 37 64 88


10 4 23 28 49 37 64 88

10 4 23 28 49 37 64 88

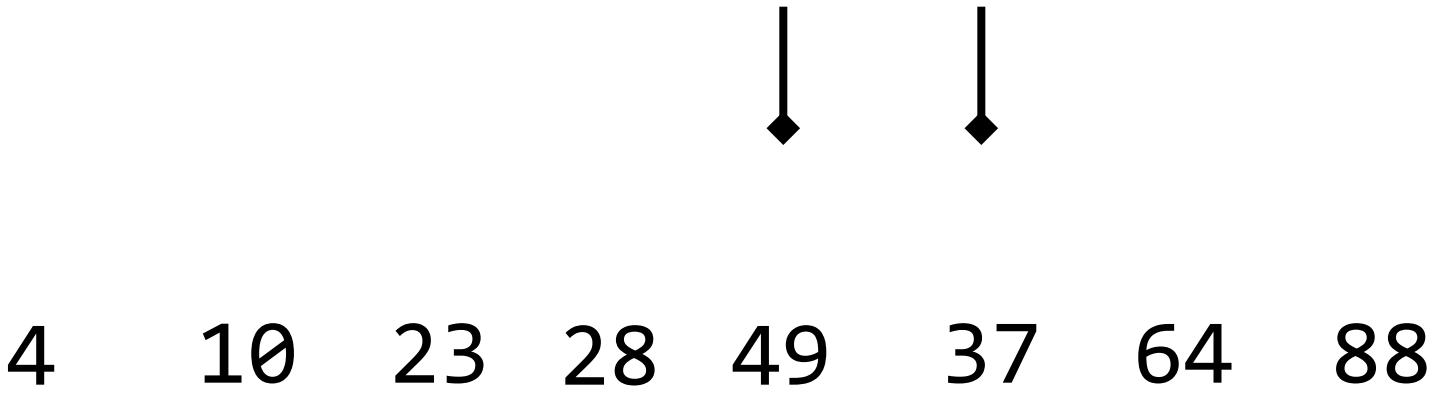
10 4 23 28 49 37 64 88

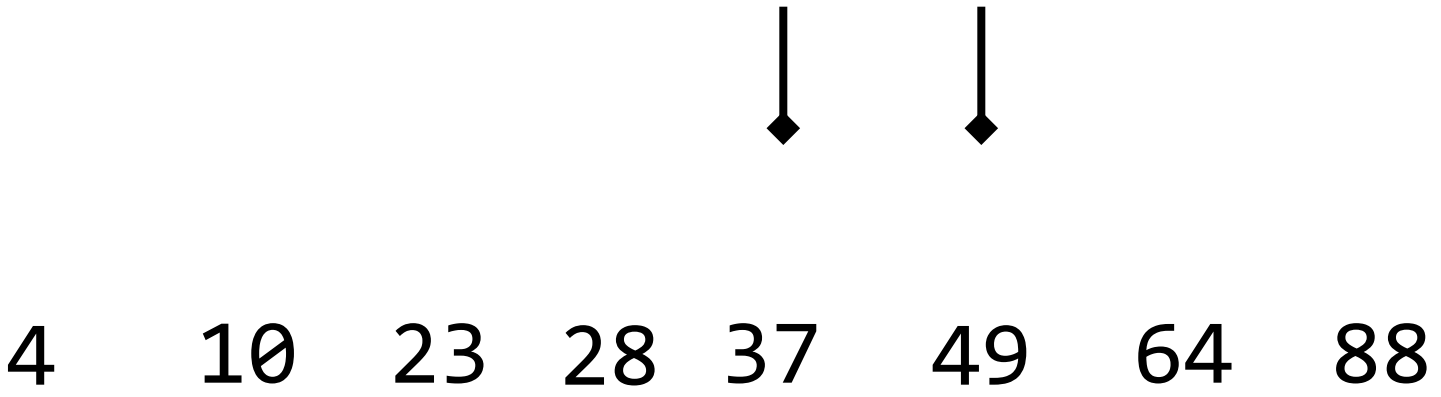
4 10 23 28 49 37 64 88

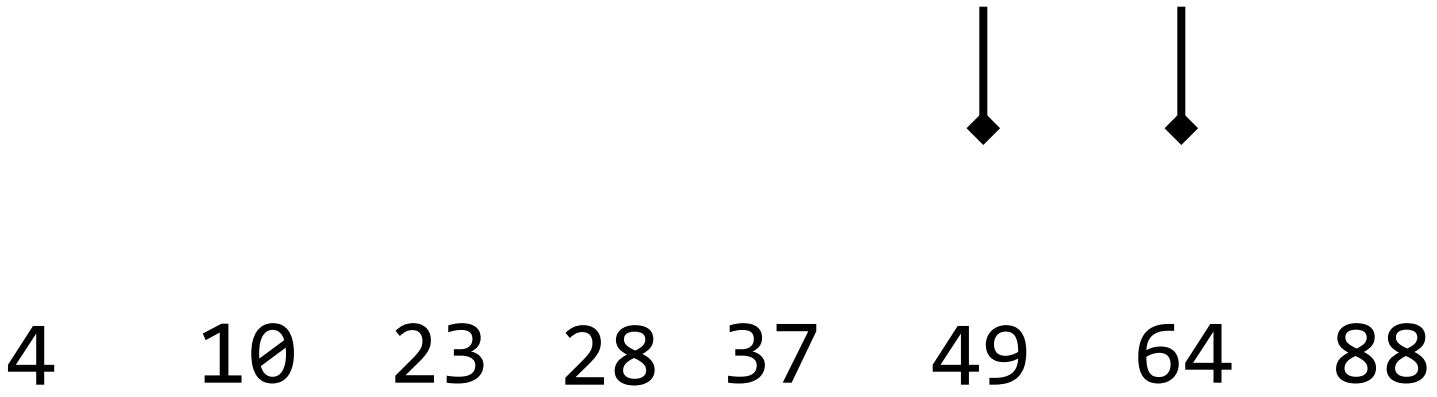
4 10 23 28 49 37 64 88

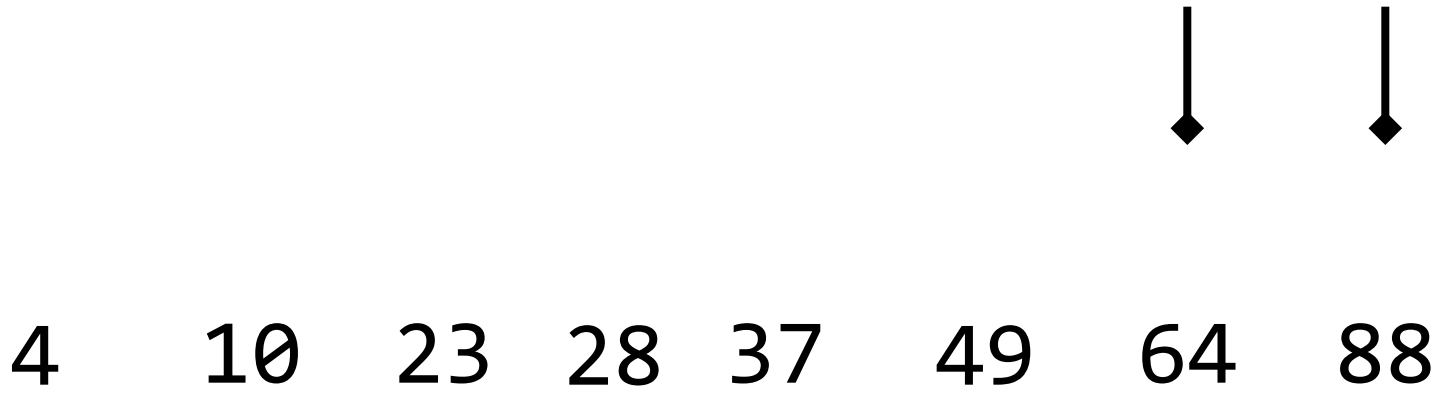
4 10 23 28 49 37 64 88

4 10 23 28 49 37 64 88


4 10 23 28 49 37 64 88


4 10 23 28 37 49 64 88


4 10 23 28 37 49 64 88


4 10 23 28 37 49 64 88

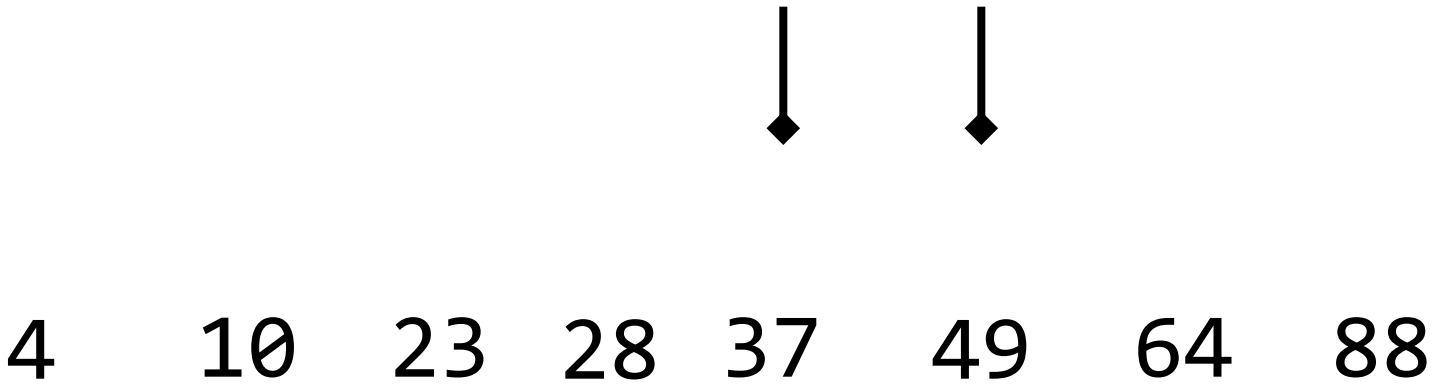
4 10 23 28 37 49 64 88

4 10 23 28 37 49 64 88

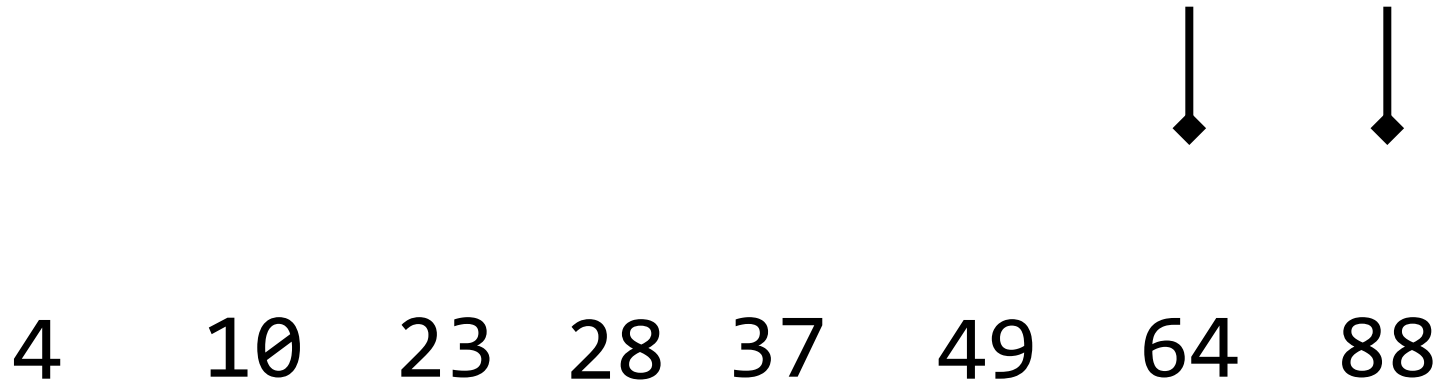
4 10 23 28 37 49 64 88

4 10 23 28 37 49 64 88


4 10 23 28 37 49 64 88


4 10 23 28 37 49 64 88


4 10 23 28 37 49 64 88


4 10 23 28 37 49 64 88

4 10 23 28 37 49 64 88

```
void bubbleSort(int list[], int n)
{
    bool isSorted = false;

    while (!isSorted)
    {
        isSorted = true;
        for (int i = 0; i < n - 1; i++)
        {
            if (list[i] > list[i + 1])
            {
                swap(&list[i], &list[i + 1]);
                isSorted = false;
            }
        }
    }
}
```