Lab deadlines

- Group 1, Friday 22nd:
  - Lab 6 (search)
  - Lab 5 deadline
- Group 2, Monday 25th:
  - Lab 6 (search)
  - Lab 5 deadline

Java tips

```
if (condition == true) { .. }
```

```
if (condition) { .. }
```

```
if (condition == false) { .. }
```

```
if (!condition) { .. }
```

```
if (condition)
  return true;
else
  return false;
```

```
return condition;
```

e.g.

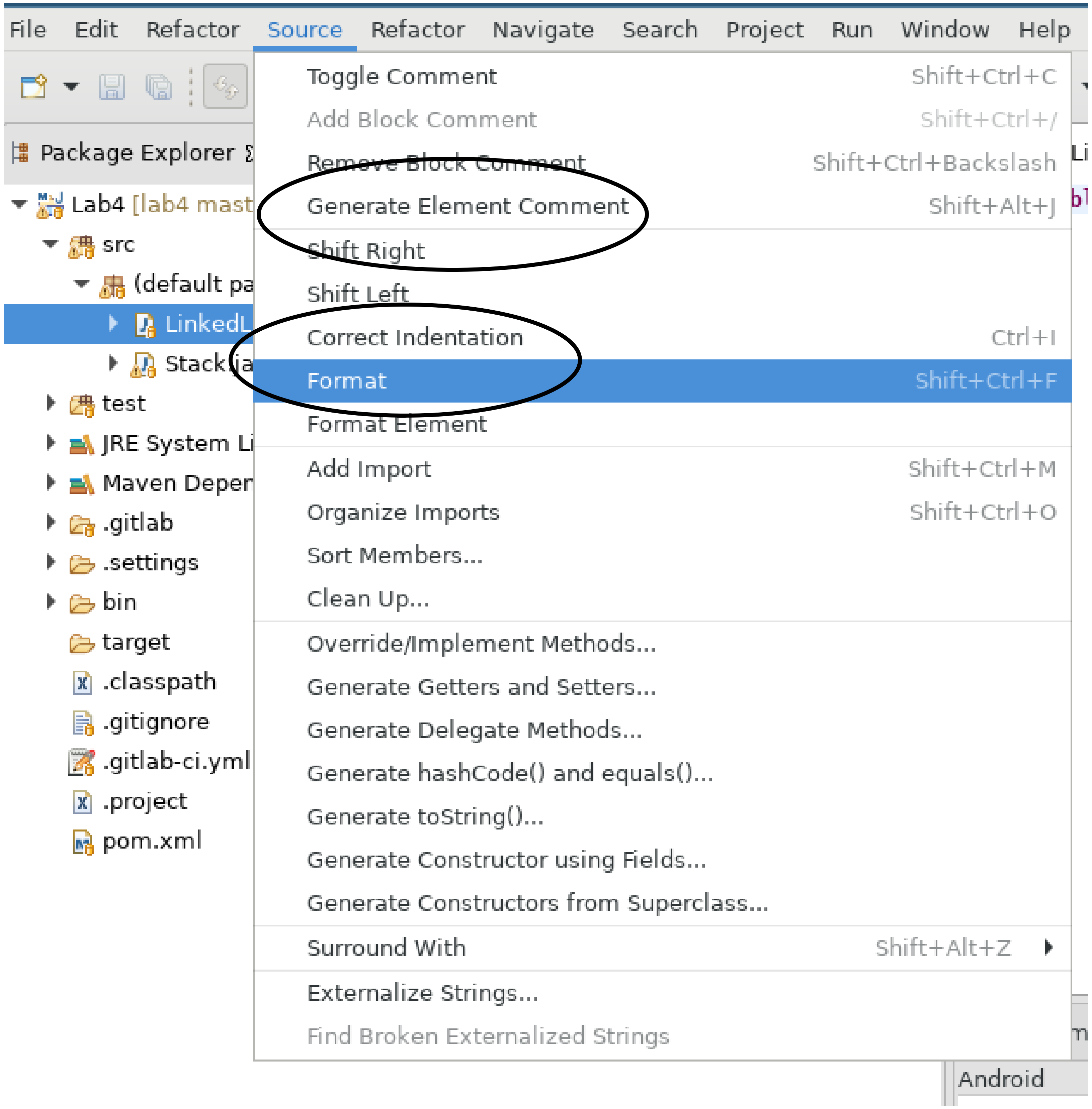
```
if (size == 0)
  return true;
else
  return false;
```

```
return (size == 0);
```

Code formatting with Java

Eclipse tooling

- Indentation
- Formatting
- Javadoc comments



Software Development 3 (F27SG)

Lecture 11

Search

Rob Stewart

Outline

By the end of the lecture you should

- understand linear search
- understand the binary search method
- be familiar with the the logarithmic function in Big-O notation: O(log N)
- be able to analyse and compare
  - linear search
  - binary search

The Importance of Search

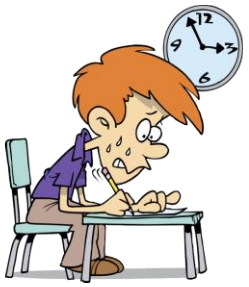
- Available data is growing faster and faster
  - internet has billion(s) of users
  - Facebook, Twitter, Instagram...
- Computers spend a lot of time just searching and sorting
- Thus, the data-structures and algorithms used are getting more and more important
  - We have already mentioned Google’s **PageRank** (search) algorithm...

Linear search demo

Linear Search (1)

- In **linear search** we
  - start at the beginning of the array
    - and compare until we find a match
    - which we return
  - Here, is a variant which searches a list of integer and returns
    - **true** if the list contains the element
    - **false** if not

```
public static boolean searchForNumber(int[] arr,int number){
  for(int i = 0; i < arr.length; i++){
    if(arr[i] == number)
      return true;
  }
  return false;
}
```



Exercise

- Given T = {0,2,3}
  1. **bool b1 = searchForNumber(T,2);**
  2. **bool b2 = searchForNumber(T,1);**

```
public static boolean searchForNumber(int[] arr,int number){
  for(int i = 0; i < arr.length; i++){
    if(arr[i] == number)
      return true;
  }
  return false;
}
```

www.socrative.com / `student login' / room: SD32018

Solution

- Given T = {0,2,3}

1. **bool v = searchForNumber(T,2);**
    - start at index 0: T[0]: T[0] != 2
    - go to next index (1): T[1] == 2: return true
2. **bool v = searchForNumber(T,1);**
    - start at index 0: T[0] != 1
    - go to next index (1): T[1] != 1:
    - go to next index (2): T[2] != 1:
    - end of array reached; return false

```
public static boolean searchForNumber(int[] arr,int number){
  for(int i = 0; i < arr.length; i++){
    if(arr[i] == number)
      return true; }
  return false;}
```

Linear Search (2)

- Typically, we search for a **key** in a collection of elements
  - return the element if a match is found
  - and throw an **exception** if failed
    - sometimes a dummy value is returned instead
- Eg. search for a **Person** (lab 1) with a given age
  - note that there may be many matches, and this will return the first

```
public Person linearSearch(Person[] arr,int age)
                                         throws NotFoundException{
 for(int i = 0; i < arr.length; i++){
   if(arr[i].getAge() == age)
     return arr[i];
 }
 throw new NotFoundException ("No person of age: " + age);
}
```

Linear Search (2)

- Typically, we search for a **key** in a collection of elements
  - return the element if a match is found
  - and throw an exception if failed
    - sometimes a **dummy value is returned instead**
- Eg. search for a **Person** (lab 1) with a given age
  - note that there may be many matches, and this will return the first

```
public Person linearSearch(Person[] arr,int age){
 for(int i = 0; i < arr.length; i++){
   if(arr[i].getAge() == age)
     return arr[i];
 }
 return null;
}
```

Linear Search of Linked Lists

- The algorithm for linked lists is the same.
  - Start at the headNode
  - Compare until the right is found
    - Next element is found by following the next reference

Linear Search of Linked Lists

- Here is the **searchForNumber** of a Linked list

```
public  boolean searchForNumber(int number){
  Node curr = headNode;
  while (curr != null){
     if(curr.getValue() == number)
        return true;
     curr = curr.getNextNode();
  }
  return false;
}
```

```
class Node {
  private int value;
  private Node nextNode;

   public int getValue() {
     return value;
   }
  public Node getNextNode() {
     return nextNode;
  }
...
}
```

Analysis of Linear Search

- We see that *linear search* iterates through the entire array
  - the number of primitives increases *linearly*
  - hence the big-O growth rate is O(n)
- What if we search a phonebook
  - with all numbers in the world...

- For ***ordered data…*** **binary search** is much more efficient
  - but this only work when the list is ***sorted****.*

- Binary search is best described via recursion

Binary Search

- We want to search in a **sorted list**
- Remember that linear search is O(n)
  - fine when n is small
  - but we can do much better...

Binary search demo

Binary Search

- **Binary search** starts in the middle
  1. if we get a match the value is returned
  2. if key searched for is greater than middle we search in the top half
  3. if key searched for is smaller than middle we search in the bottom half
  4. if we end up with an empty list
     1. i.e. no element left in the part we are searching
     2. then search has failed!

Binary Search example

- Given an array A, we want to apply binary search to find 4

0	2	4	6	8	10	12	14

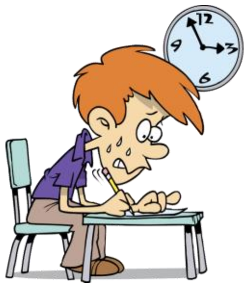
- We start in the middle, i.e. index 3, where A[3] =6
- As 4 < 6, 4(if it exists) will be on the left of index 3,
  - i.e. between index 0 and index 2

0	2	4	6	8	10	12	14

- We then find the middle of 0 and 2, which is 1
- As A[1] = 2 < 4, we know that 4 (if it exists), is on the right of index 1
  - We already know that it is on left of index 3
  - Meaning it will be between index 2 and 2, i.e. at index 2

0	2	4	6	8	10	12	14

- We then find the middle of index 2 and 2, which is 2
- **As A[2] = 4 we are finished!**



Exercise

- Given an array A

0	2	4	6	8	10	12	14

1. Use binary search to find 2
2. Use binary search to find 1

- In both cases, show
  - What the middle index is, and which value this index has
  - The parts of the array you have identified that the value will be
    - i.e. which index is must be between

Start in the middle

- if we get a match the value is returned
- if the key searched for is greater than the middle we search in the top half
- if the key searched for is smaller than the middle we search in the bottom half
- if we end up with an empty list search has failed!

The Binary Search Method method signature

- We assume we have an array **arr** of **Person** object *ordered by their age*
- Given an **age** we would like to return a **Person** object of that age
- What are the inputs, returns value and error conditions?
- Inputs:
  - **int age** to search for
  - the **first** index of **arr** we should search in
  - the **last** index of **arr** we should search in
- Returns: a **Person** object
- Error conditions: a person of that **age** is not present in **arr**

```
public Person binarySearch(int age,int first,int last)
                                        throws NotFoundException{
  ...
}
```

The Binary Search Method
base cases

- What are the base cases?
  - failure case
  - success case

```
public Person binarySearch(int age,int first,int last)
                                                    throws NotFoundException{
  if (first > last) // failure case
     throw new NotFoundException ("element not found");
  else {
    int middle = (first + last) / 2;
    int middle_age = arr[middle].getAge();
    if (middle_age == age) // success case
         return arr[middle];
    ...
  }
}
```

The Binary Search Method step cases

- What are the step cases?
  - smaller than means search bottom half
  - greater than means search top half

```
public Person binarySearch(int age,int first,int last)
                                          throws NotFoundException{   ...
  if (middle_age == age) // success case
      return arr[middle];
  else if (age < middle_age) // bottom half
     return binarySearch(age,first,middle-1) ;
  else // top half
     return binarySearch(age,middle+1,last) ;
 }
}
```

The Binary Search Method

```
public Person binarySearch(int age,int first,int last)
                                                throws NotFoundException{
  if (first > last) // failure case
      throw new NotFoundException ("element not found");
  else {
    int middle = (first + last) / 2;
    int middle_age = arr[middle].getAge();
    if (middle_age == age) // success case
         return arr[middle];
  else if (age < middle_age) // bottom half
       return binarySearch(age,first,middle-1) ;
   else // top half
       return binarySearch(age,middle+1,last) ;
  }
}
```

Example

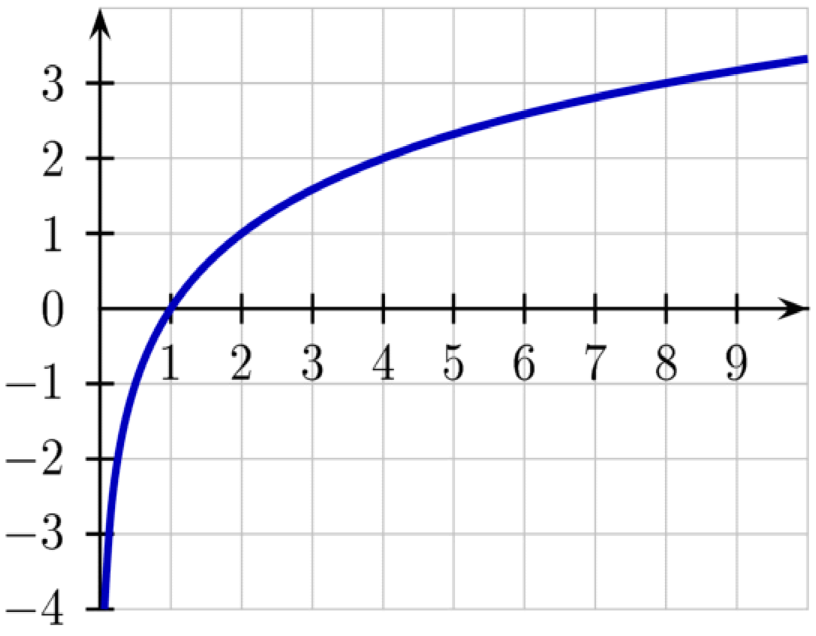
- Let's assume we have the following array **arr:**

```
Person  p1 = new Person("Isobel","Gordon",9);
Person  p2 = new Person("Montgomery", "Ross", 19);
Person  p3 = new Person("Janneth","Johnston",23);
Person  p4 = new Person("Jackie","Morrison", 39);
Person  p5 = new Person("Catriona", "Miller",49);
Person[] arr = {p1,p2,p3,p4,p5};
```

- Draw the recursive trace of **binarySearch(19,0,4)**

1. middle = (4+0)/2 = 2, arr[2].getAge() = 23 > 19
   call bottom half: **binarySearch(19,0,1)**
2. middle = (1+0)/2 = 0, arr[20].getAge() = 9 < 19
   call top half: **binarySearch(19,1,1)**
3. middle = (1+1)/2 = 1, arr[1].getAge() = 19 = 19
   return arr[1]

Big-O: the Logarithmic Function O(log N)



- To illustrate
  - log 8 = 3 since $2^3 = 8$

- when **doubling the size of N, *the logarithmic function increase by 1***

Big-O: the Logarithmic Function O(log N)

binary logarithm *log n*

– "the power in which you have to raise 2 to get number **n**"

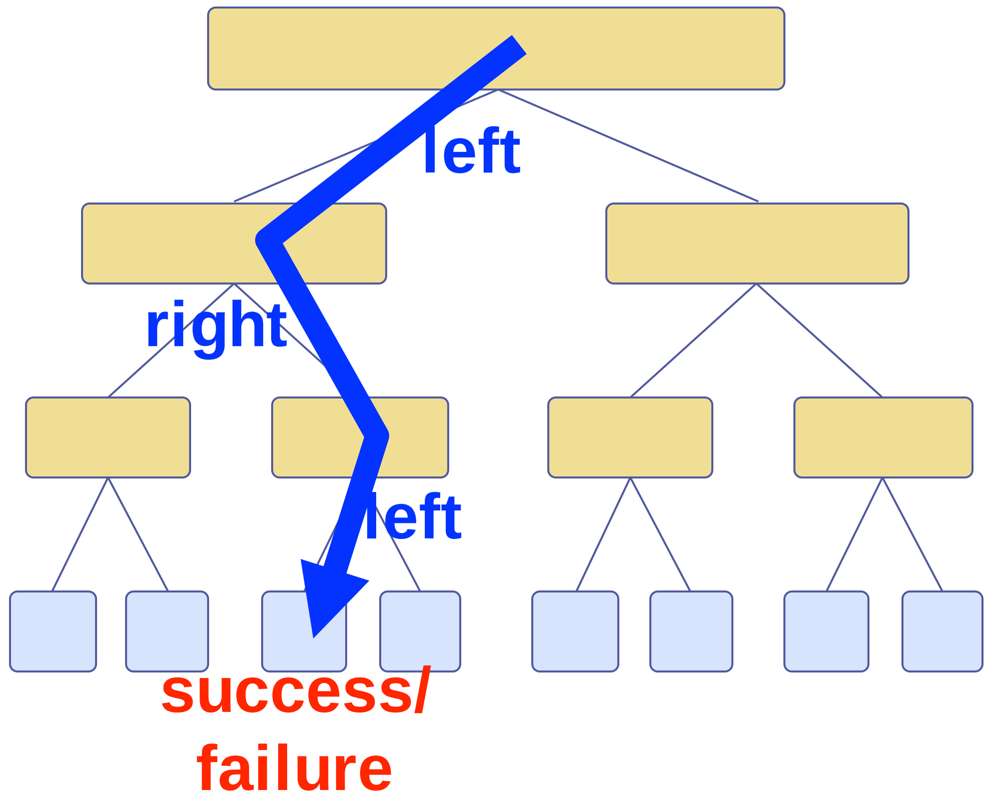
- It is the inverse of function $f(n) = 2^n$
  – $\log 2 = x \Leftrightarrow 2^x = 2 \;= 2 = 2^1 \Rightarrow x=1$
  – $\text{Log } 4 = x \Leftrightarrow 2^x = 4 \;= 2*2 = 2^2 \Rightarrow x=2$
  – $\log 8 = x \Leftrightarrow 2^x = 8 \;= 2*2*2= 2^3 \Rightarrow x=3$
  – $\log 16 = x \Leftrightarrow 2^x = 16 \;= 2*2*2*2= 2^4 \Rightarrow x=4$
  – $\log 32 = x \Leftrightarrow 2^x = 32 \;= 2*2*2*2*2= 2^5 \Rightarrow x=5$

Analysis of Binary Search

- For each step in b*inary search*, we either return the value
  - or search the left or right, where size of the array is halved

- We can therefore draw the *search space*
  - possible paths we can take

  as a *binary tree*
  - each node is half the size of the parent
- Each step halves size of array
- .. stop when there is only one element
- .. hence the *height of the tree is O(log n)*
- *height is actually log n + 1*
  - *but lesser terms are removed in Big-O*



Analysis of Binary Search

- In the worst case
  - the list is halved until there is only one element left
- This means we have reached the bottom of the tree
- When traversing from the root to the bottom
  - we have made **O(log N) (recursive) calls**.
  - The height of the tree is O(log N)
- For each node in the tree
  - We perform constant number of operations O(1)
- Thus, binary search has a **logarithmic** growth rate:
  - O(log N) + O(1)

  => O(log N) ***(simplify Big-Oh formula)***

Example

- Array of size 8 (n == 8)
  - First search on array of 8 elements
  - 1st **recursive** call: 4
  - 2nd **recursive** call: 2
  - 3rd **recursive** call: 1 (terminates)
  - i.e. for height of 4

  => 3 + 1

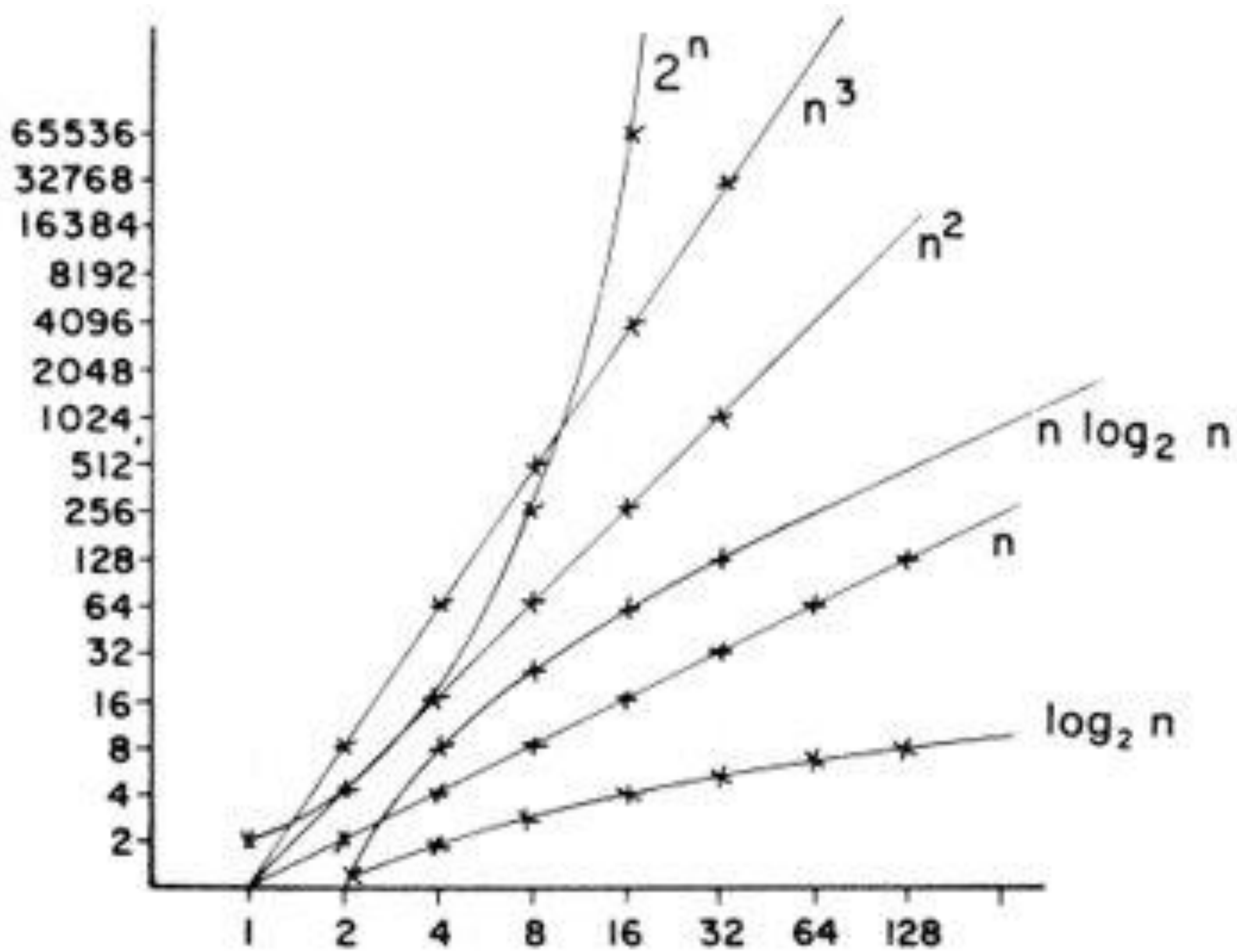
  => log 8 + 1 ***(rewrite 3 => log 8)***

  => log n + 1 ***(since n=8)***

  => log n ***(simplify Big-Oh formula)***

Analysis of Binary Search

- Logarithm has a smaller growth rate compared with linear functions
- Meaning *binary search is* ***faster*** *than linear search*



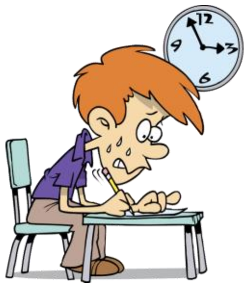
Binary Search of Linked List?

- Why cannot we do the same for linked lists?

Binary Search of Linked List?

- Why cannot we do the same for linked lists?
  - No random access
    - We have to iterative to the middle element
    - Meaning O(N)…

Search Space Race



Exercise

Assume array **arr** has the following elements:

int[] arr = {2,4,6,8,10,12,14,16};

Draw the recursion trace of the algorithm with the following calls:

(a) **binarySearch(16,0,7);**

(b) **binarySearch(15,0,7);**

For each call describe

- the value of each parameter
- what the **middle**

```
public int binarySearch (int key, int first, int last)
  throws NotFoundException {
    if (first > last)
      throw new NotFoundException ("not found");
    else {
      int middle = (first + last) / 2;
      if (key == arr[middle])
         return arr[middle];
      else if (key < arr[middle])
         return binarySearch (key, first, middle-1);
       else
         return binarySearch (key, middle+1, last);
  }
}
```

e.g.

binarySearch(16, 0,7);

middle = ?

binarySearch(?, ?, ?);

middle = ?

e.t.c…

Summary

- Linear search
- Binary search (using recursion)
- The logarithmic function in Big-O: O(log N)
- Compared running time
  - linear search vs binary search
- Homework: exercises in these slides (Vision)
- Attendance sheet
- Tomorrow: **trees**