CS 2050
Computer Science II

Lesson 08

Metropolitan State University of Denver
Lives Transformed

Agenda

- Search Algorithms:
  - Linear Search
  - Binary Search
- Big O Notation

Search Algorithms

- A method of finding an item (or group of items) with specific properties within a collection of items (search space)
- The collection of items is normally represented using an array or a list

Linear Search

- In a linear search, each of the items in the collection is compared to see if it is the item searched for

Linear Search

```
static boolean lsearch(int data[], int el) {
   for (int i = 0; i < data.length; i++)
       if (data[i] == el)
           return true;
   return false;
}
```

Linear Search

- The linear search can be optimized if the collection is sorted

Linear Search

```
static boolean lsearch2(int data[], int el) {
   for (int i = 0; i < data.length; i++) {
       if (data[i] == el)
           return true;
       if (data[i] > el)
           break;
   }
   return false;
}
```

Binary Search

- The binary search assumes that the collection is sorted
- The item in the middle of the collection is compared to the element searched for
- If the element is not the middle item, the search prossegues to the left/right, depending whether the element is less/greater than the middle item

Binary Search

```java
static boolean bsearch(int data[], int start, int end, int el) {

    // base case #1 (parameter validation)
    if (start > end || start < 0 || end < 0 || start >= data.length || end >= data.length)
        return false;

    // get the middle element
    int middle = (start + end) / 2;  // or start + (end - start) / 2

    // is middle the element we are searching for?
    if (data[middle] == el)
        return true;

    // base case #2
    if (start == end)
        return false;

    // is the element greater than middle -> go right then
    if (el > data[middle])
        return bsearch(data, start: middle + 1, end, el);

    // if not, go left
    return bsearch(data, start, end: middle - 1, el);
}
```

Binary Search

```
el = 12
```

```
1 3 10 12 13 27 34 56 77 79 84 95
```

Binary Search

`el = 12`

`1 3 10 12 13 27 34 56 77 79 84 95`

`start = 0`

`end = 11`

Metropolitan State University of Denver
Lives Transformed

Binary Search

`el = 12`

`1 3 10 12 13 27 34 56 77 79 84 95`

`start = 0` `middle = 5` `end = 11`

Binary Search

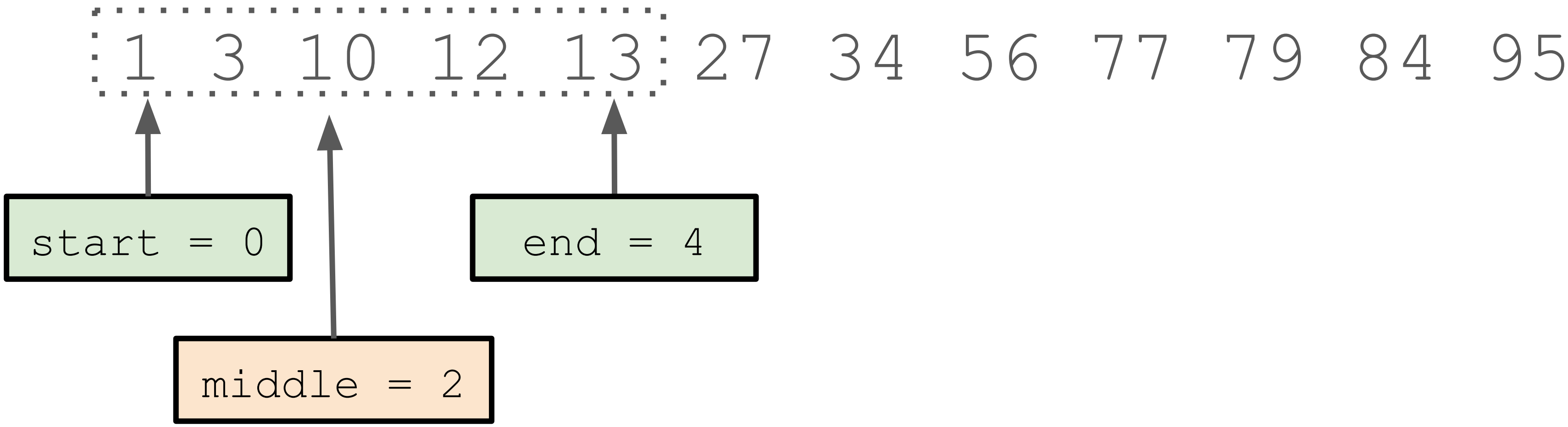
`el = 12`

```
1 3 10 12 13 27 34 56 77 79 84 95
```

`start = 0` `end = 4`

Binary Search

el = 12



METROPOLITAN STATE UNIVERSITY OF DENVER
LIVES TRANSFORMED

Binary Search

`el = 12`

```
1 3 10 12 13 27 34 56 77 79 84 95
```

`end = 4`

`start = 3`

Binary Search

`el = 12`

`1 3 10 12 13 27 34 56 77 79 84 95`

`end = 4`

`start = 3`

`middle = 3`

Binary Search

```
el = 80
```

```
1  3  10  12  13  27  34  56  77  79  84  95
```

Binary Search

`el = 80`

`1 3 10 12 13 27 34 56 77 79 84 95`

`start = 0`

`end = 11`

Binary Search

el = 80

1 3 10 12 13 27 34 56 77 79 84 95

start = 0

middle = 5

end = 11

Binary Search

```
el = 80
```

```
1 3 10 12 13 27 34 56 77 79 84 95
```

```
start = 6
end = 11
```

Binary Search

el = 80

1 3 10 12 13 27 34 56 77 79 84 95

start = 6

end = 11

middle = 8

METROPOLITAN STATE UNIVERSITY℠ OF DENVER

LIVES TRANSFORMED

Binary Search

el = 80

1 3 10 12 13 27 34 56 77 79 84 95

end = 11

start = 9

Binary Search

`el = 80`

`1 3 10 12 13 27 34 56 77 79 84 95`

`end = 11`

`start = 9`

`middle = 10`

Binary Search

```
el = 80
```

```
1 3 10 12 13 27 34 56 77 79 84 95
```

```
start =
end = 9
```

Binary Search

`el = 80`

`1 3 10 12 13 27 34 56 77 79 84 95`

```
start =
end = 9
```

Big O Notation

Imagine you want to know how long it will take for the following program to execute.

```
static int sum(int x) {
  int s = 0;
  for (int i = 1; i <= x; i++)
    s += i;
  return s;
}
```

After reflecting on the problem you conclude that many factors may influence your answer, such as:

- the value of the input parameter x,
- the speed of the computer on which it is run, and
- the efficiency of your Java implementation on that particular computer.

Big O Notation

```
int s = 0; // 1 step
int i = 1; // 1 step
i <= x;    // x steps
i++        // x steps
s += i;    // x steps
return s;  // 1 step
```

Therefore, the total number of steps of our program is given by the formula:

```
1 + 1 + x + x + x + 1 = 3x + 3
```

Big O Notation

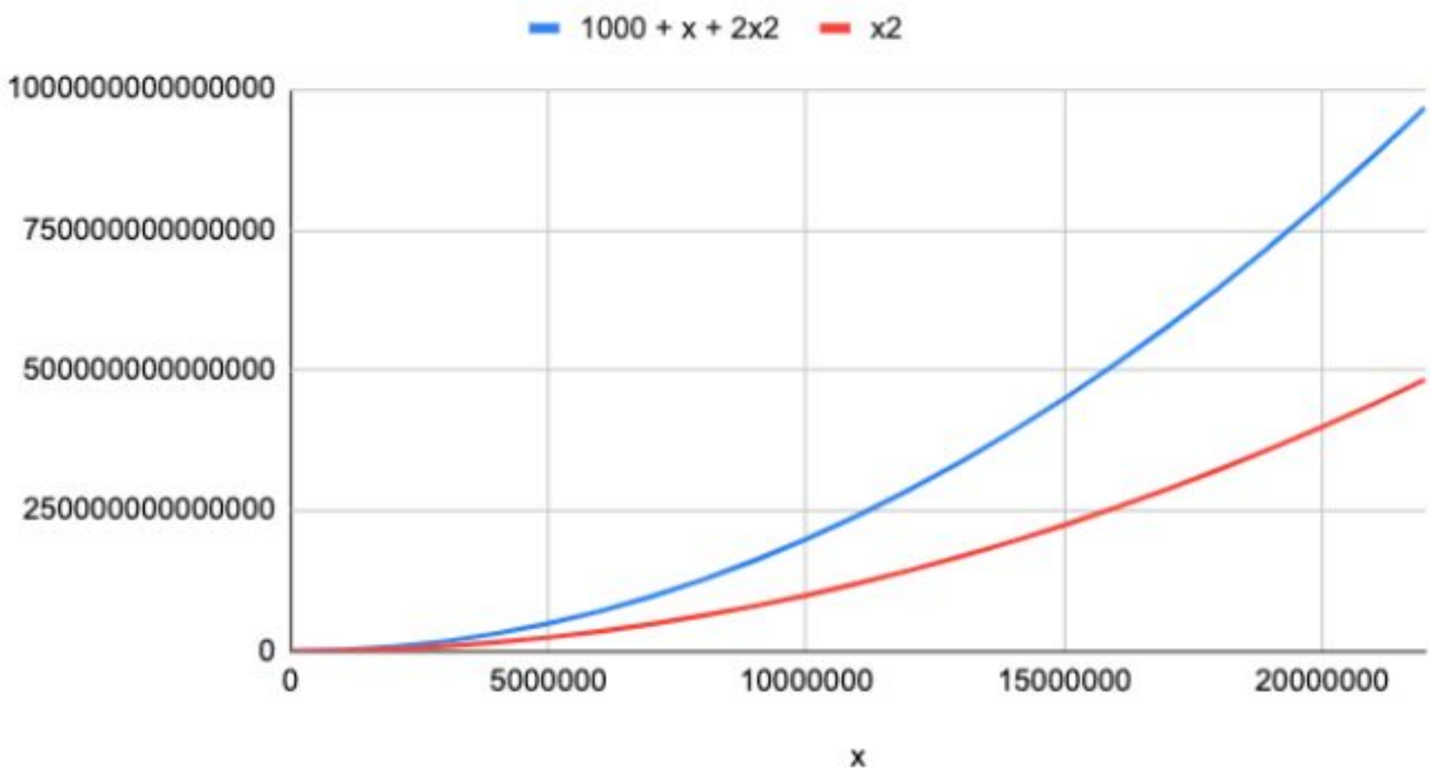
Now let's evaluate the execution time of the following program.

```
static int f(int x) {
  int steps = 1;

  // a loop that takes a constant number of steps
  for (int i = 0; i < 1000; i++)
    steps++;

  // a loop that takes x number of steps
  for (int i = 0; i < x; i++)
      steps++;

  // a loop that takes 2x2 number of steps
  for (int i = 0; i < x; i++)
      for (int j = 0; j < x; j++) {
          steps++;
          steps++;
      }

  // return the number of steps
  return steps;
}
```

The running time of function f can be described as:

```
3005 + 6x + 4x^2
```

Big O Notation



Big O Notation

O(1): constant running time (it does not depend on the input's size)

O(log x): logarithmic running time (remember that log x < x)

O(x): linear running time

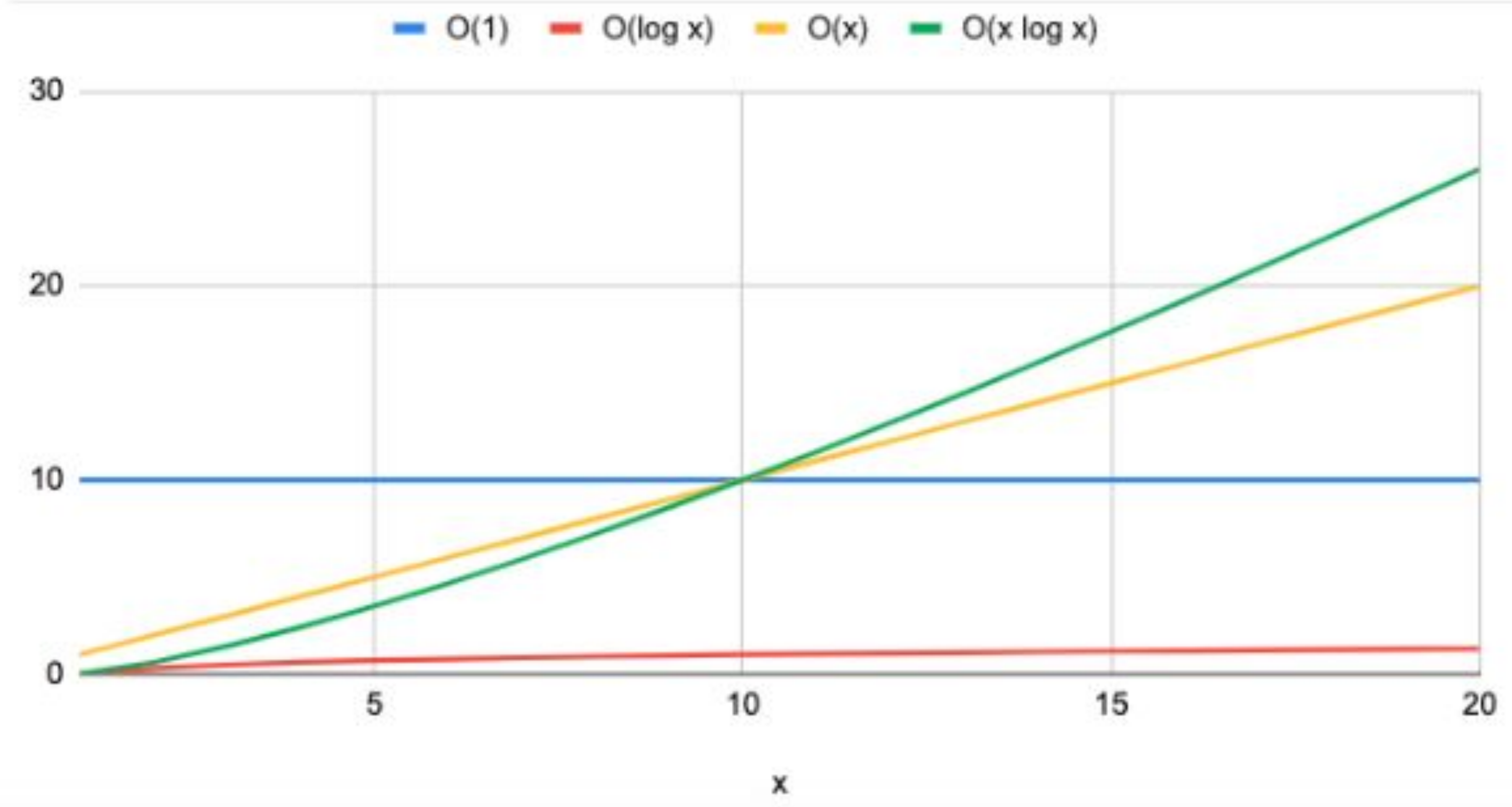
O(x log x): log-linear running time (remember that x log x < x2)

O(x^k): polynomial running time (notice that k is a constant)

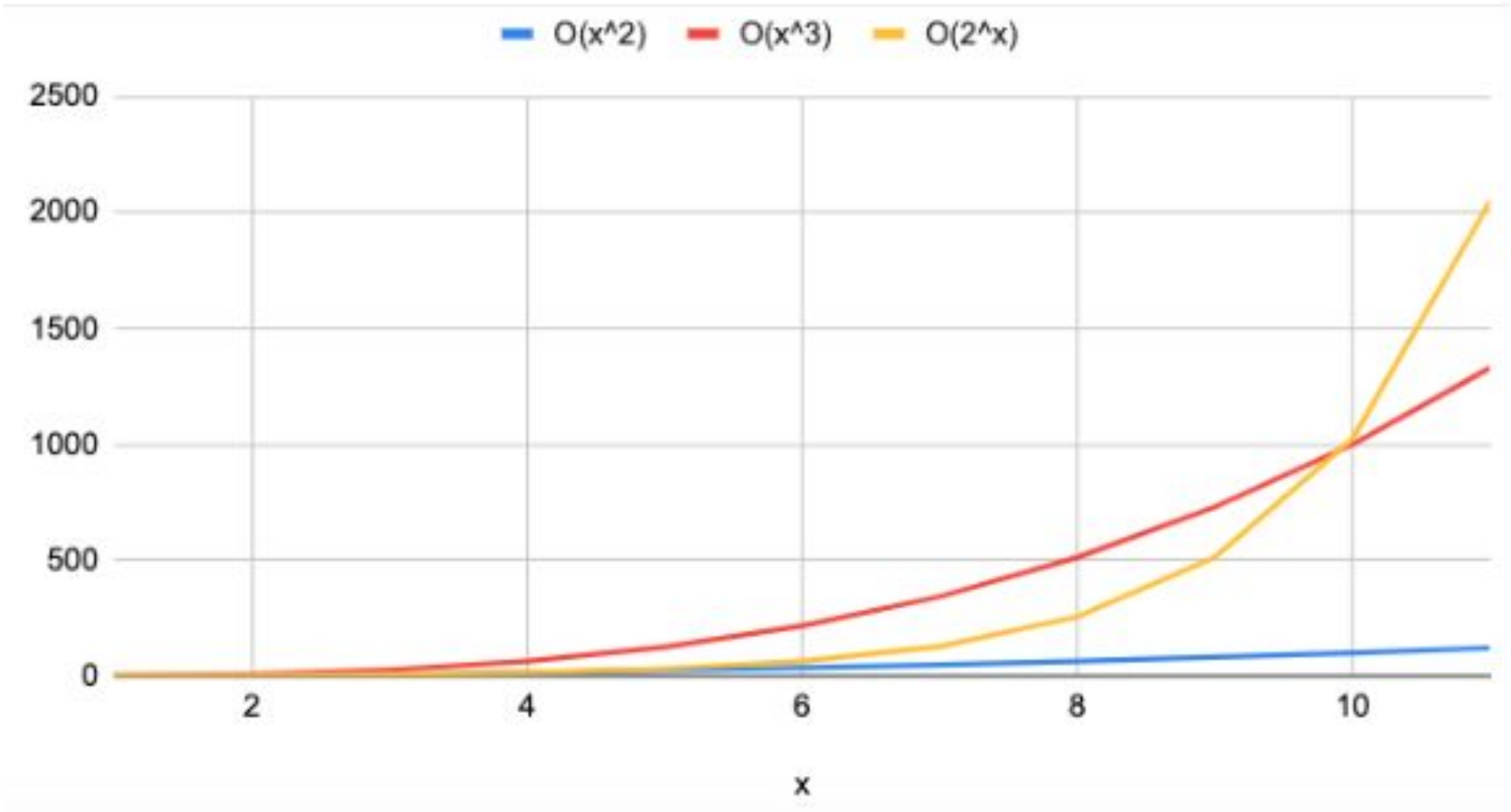
O(c^x): exponential running time (notice that c is a constant raised to a power based on the input's size)

The figures below illustrate the asymptotic growth of those running times for comparison.

Big O Notation



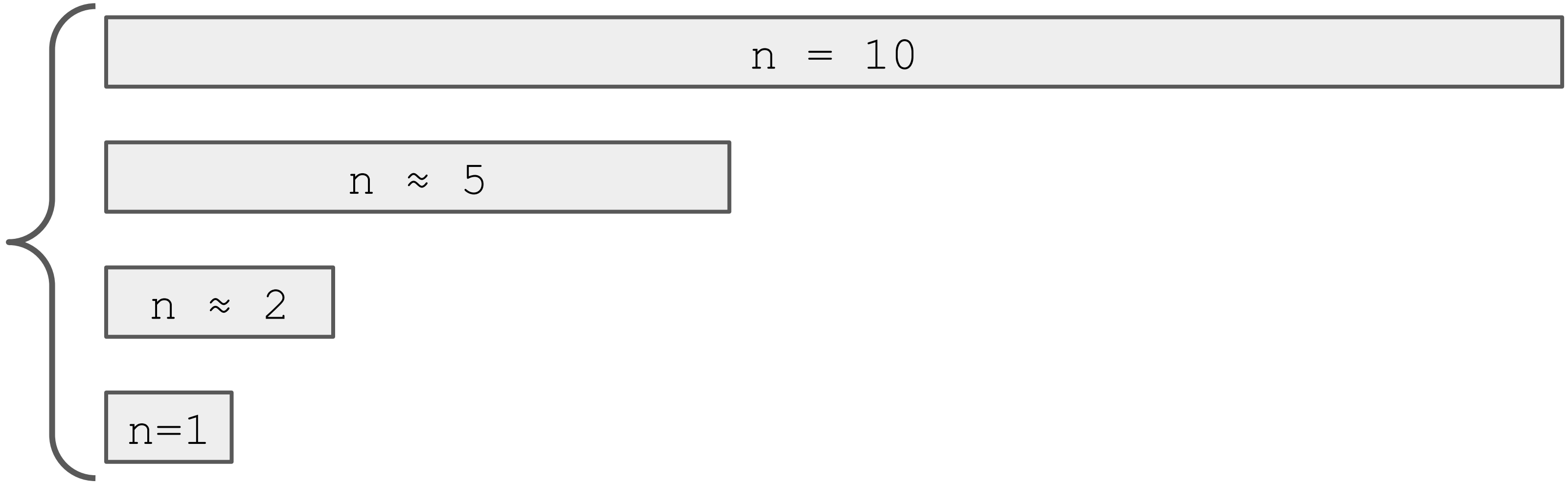
Big O Notation



Binary Search (complexity)

$\log_2(n=10)$

$= 3.32$

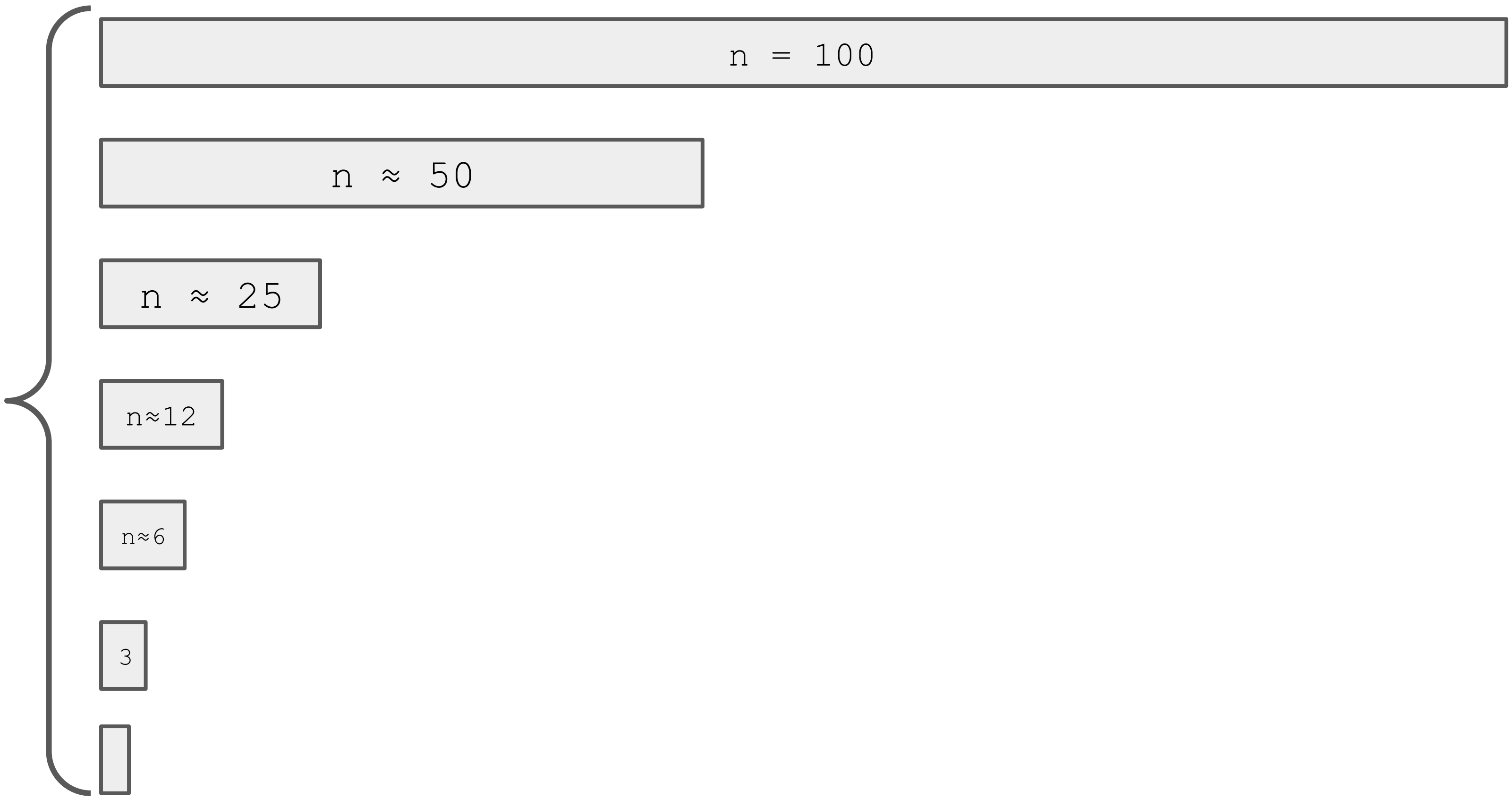


METROPOLITAN STATE UNIVERSITY OF DENVER
LIVES TRANSFORMED

Binary Search (complexity)

$\log_2(n=100)$

$= 6.64$



METROPOLITAN STATE UNIVERSITY℠ OF DENVER

LIVES TRANSFORMED