Algorithms

Part b: Algorithm Running Times

Ian Ludden

Learning Objectives

By the end of this lesson, you will be able to:

Learning Objectives

By the end of this lesson, you will be able to:

- Be familiar with the overall structure and big-O running times of some representative algorithms.

Learning Objectives

By the end of this lesson, you will be able to:

- Be familiar with the overall structure and big-O running times of some representative algorithms.
- Given a recursive algorithm (familiar or unfamiliar), express its running time as a recursive definition.

Binary Search (Iterative)

Given a *sorted* array of n integers, check whether it contains a given value.

```
1  function binary_search(arr, x)
2      lower_bound = 1
3      upper_bound = arr.length
4
5      while lower_bound <= upper_bound
6          middle = floor((lower_bound + upper_bound) / 2)
7
8          if arr[middle] = x
9              return middle
10         else if arr[middle] < x
11             lower_bound = middle + 1
12         else    // arr[middle] > x
13             upper_bound = middle - 1
14
15     return -1   // x is not in arr
```

Game: I pick a number between 1 and 100 (incl.)
You try to guess.
I tell you:
a) correct
b) too high
c) too low

1 25 31 37 75 100

$50 = \left\lfloor \frac{1+100}{2} \right\rfloor$

$\log_2(\text{upper} - \text{lower}) + 1$

Binary Search (Iterative)

Given a *sorted* array of n integers, check whether it contains a given value.

```
1  function binary_search(arr, x)
2      lower_bound = 1
3      upper_bound = arr.length
4
5      while lower_bound <= upper_bound
6          middle = floor((lower_bound + upper_bound) / 2)
7
8          if arr[middle] = x
9              return middle
10         else if arr[middle] < x
11             lower_bound = middle + 1
12         else    // arr[middle] > x
13             upper_bound = middle - 1
14
15     return -1   // x is not in arr
```

O(1)

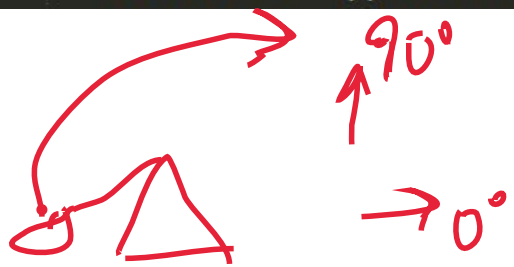
O(1)

how many iterations?

$O(\log_2 n) = O(\log n)$

$O(\log_5 n)$

$\log_2 n = \log_5 n \cdot \log 5$



target

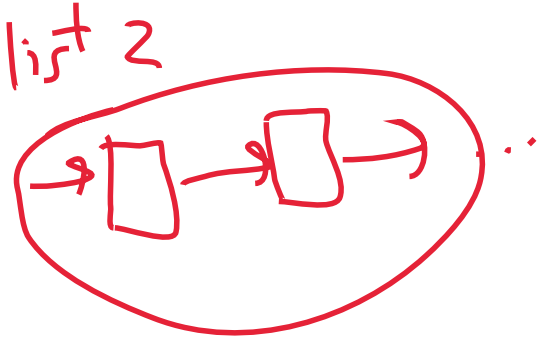
Merge Two Lists (Recursive)

Given two *sorted* lists of real numbers, merge them into one sorted list.

$n = n_1 + n_2$

```
1   function merge(list1, list2)
2       if list1 is empty
3           return list2
4       if list2 is empty
5           return list1
6
7       if first(list1) <= first(list2)
8           return cons(first(list1), merge(rest(list1), list2))
9       else
10          return cons(first(list2), merge(list1, rest(list2)))
11
12
```

c



$T(1) = c$

$T(n) = T(n-1) + d$

$T(n) = nd + c$, which is $O(n)$.

Merge Two Lists (Recursive)

Given two *sorted* lists of real numbers, merge them into one sorted list.

```
1   function merge(list1, list2)
2       if list1 is empty
3           return list2
4       if list2 is empty
5           return list1
6
7       if first(list1) <= first(list2)
8           return cons(first(list1), merge(rest(list1), list2))
9       else
10          return cons(first(list2), merge(list1, rest(list2)))
11
12
```

Mergesort (Recursive)

Sort a given list of real numbers.

```
1  function mergesort(list = (A1, A2, ..., An))
2      if n = 1
3          return list
4
5      m = floor(n / 2)
6
7      lower_half = (A1, A2, ..., Am)
8      upper_half = (Am+1, Am+2, ..., An)
9
10     return merge(mergesort(lower_half), mergesort(upper_half))
11
```

Base case O(1)

split in the middle] O(n)

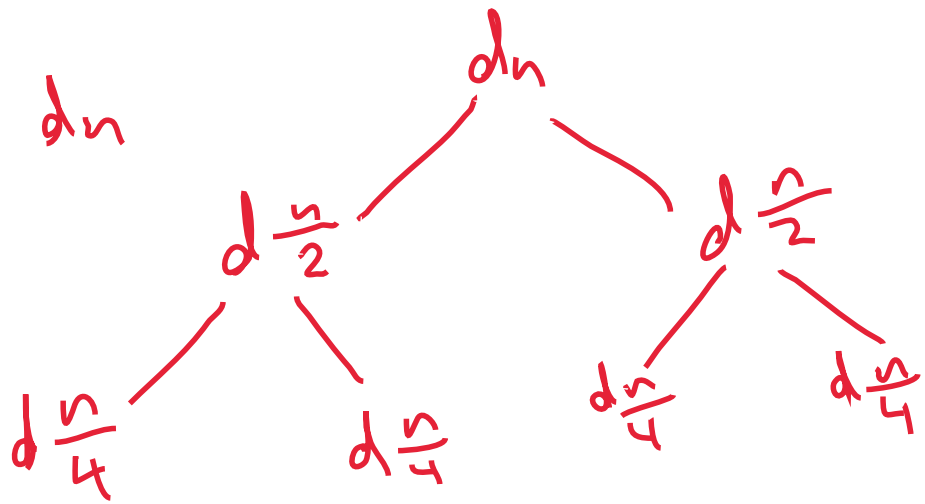
] O(n) + 2T($\frac{n}{2}$)

recursive calls

$T(1) = O(1) = c$

$T(n) = 2T(\frac{n}{2}) + O(n)$ → dn

$O(n \log n)$



$h = O(\log n)$

Mergesort (Recursive)

Sort a given list of real numbers.

```
function mergesort(list = (A1, A2, ..., An))
    if n = 1
        return list

    m = floor(n / 2)

    lower_half = (A1, A2, ..., Am)
    upper_half = (Am+1, Am+2, ..., An)

    return merge(mergesort(lower_half), mergesort(upper_half))

```

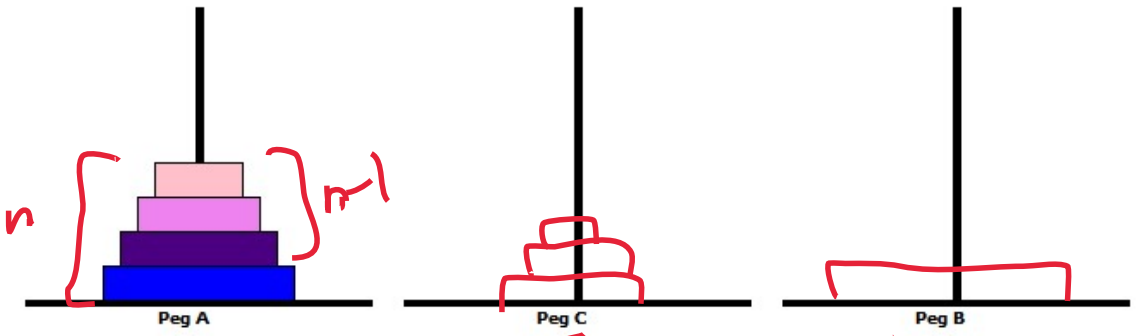
Tower of Hanoi (Recursive)

Move a tower of disks from one peg to another.
(Link to Interactive Website)

```
1 function hanoi(A, B, C, d1, d2, ..., dn)
2     if n = 1
3         move d1 from A to B
4     else
5         hanoi(A, C, B, d1, d2, ..., dn-1)
6         move dn from A to B
7         hanoi(C, B, A, d1, d2, ..., dn-1)
8
```

start, end, temp.] Base case

Tower of Hanoi



$M(1) = 1$

$M(n) = 2 \cdot M(n-1) + 1$

$M(n) = 2 \cdot (2(M(n-2))+1)+1$

$= \dots$

$= 2^k M(n-k) + \sum_{i=0}^{k-1} 2^i$

$h = n-1$, so $M(n) = 2^{n-1} M(1) + \sum_{i=0}^{n-2} 2^i \rightarrow = 2^n - 1.$

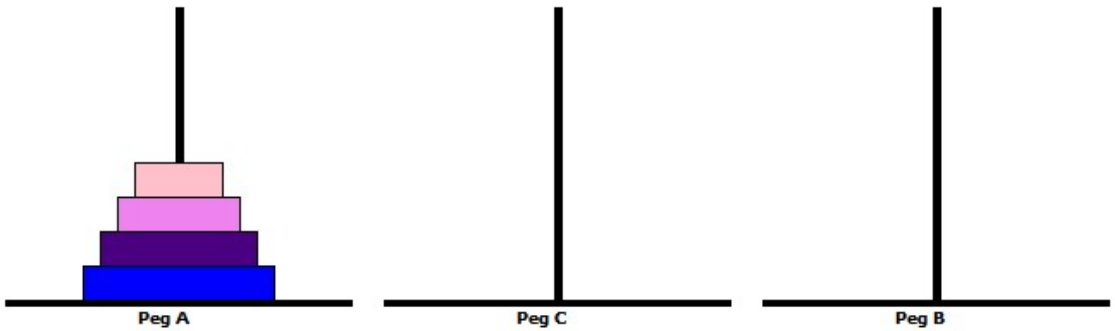
$= 2^{n-1} + 2^{n-1} - 1$

Tower of Hanoi (Recursive)

Move a tower of disks from one peg to another.
(Link to Interactive Website)

```
1 function hanoi(A, B, C, d1, d2, ..., dn)
2     if n = 1
3         move d1 from A to B
4     else
5         hanoi(A, C, B, d1, d2, ..., dn-1)
6         move dn from A to B
7         hanoi(C, B, A, d1, d2, ..., dn-1)
8
```

Tower of Hanoi



Recap: Learning Objectives

By the end of this lesson, you will be able to:

- Be familiar with the overall structure and big-O running times of some representative algorithms.
- Given a recursive algorithm (familiar or unfamiliar), express its running time as a recursive definition.