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.

```
function binary_search(arr, x)
    lower_bound = 1
    upper_bound = arr.length

    while lower_bound <= upper_bound
        middle = floor((lower_bound + upper_bound) / 2)

        if arr[middle] = x
            return middle
        else if arr[middle] < x
            lower_bound = middle + 1
        else    // arr[middle] > x
            upper_bound = middle - 1

    return -1   // x is not in arr
```

Binary Search (Iterative)

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

```
function binary_search(arr, x)
    lower_bound = 1
    upper_bound = arr.length

    while lower_bound <= upper_bound
        middle = floor((lower_bound + upper_bound) / 2)

        if arr[middle] = x
            return middle
        else if arr[middle] < x
            lower_bound = middle + 1
        else    // arr[middle] > x
            upper_bound = middle - 1

    return -1   // x is not in arr
```

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
```

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.

```
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))

```

Mergesort (Recursive)

Sort a given list of real numbers.

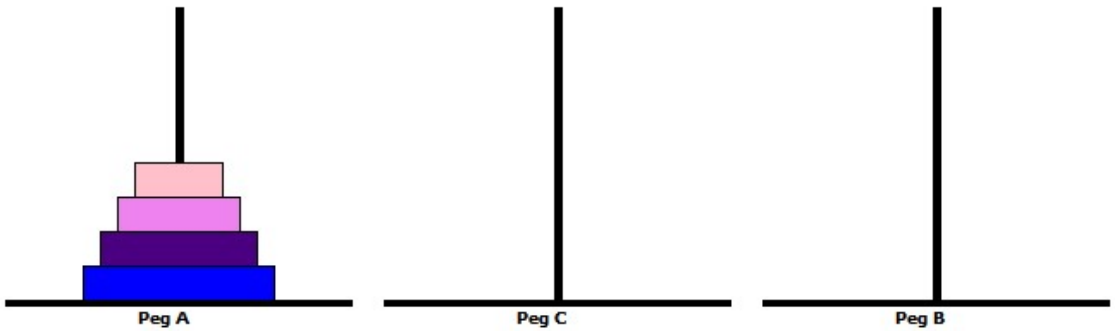
```
function mergesort(list = (A1, A2, ..., An))
    if n = 1
        return list

    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
```

Tower of Hanoi

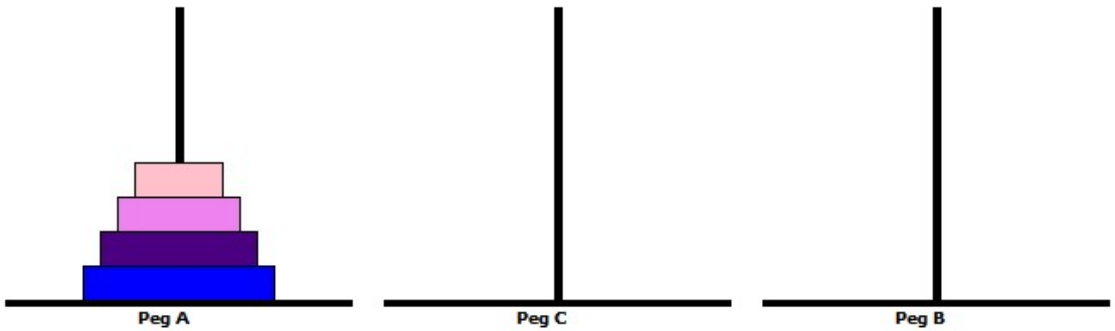


Tower of Hanoi (Recursive)

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.