Algorithms

Part a: Basic Data Structures

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 able to read code that uses basic linked list operations (first, rest, cons).

Learning Objectives

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

- Be able to read code that uses basic linked list operations (first, rest, cons).
- Know the big-O running times of basic operations on linked lists and arrays.

Learning Objectives

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

- Be able to read code that uses basic linked list operations (first, rest, cons).
- Know the big-O running times of basic operations on linked lists and arrays.
- For an algorithm involving loops (perhaps nested), express its running time using summations.

Learning Objectives

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

- Be able to read code that uses basic linked list operations (first, rest, cons).
- Know the big-O running times of basic operations on linked lists and arrays.
- For an algorithm involving loops (perhaps nested), express its running time using summations.
- Given an unfamiliar but fairly simple function in pseudo-code, analyze how long it takes using big-O notation.

Arrays vs. Linked Lists

	Arrays			**Linked Lists**		
	R/W	Add	Remove	R/W	Add	Remove
At/near start						
In middle						
At end						

Arrays vs. Linked Lists

	Arrays			**Linked Lists**		
	R/W	Add	Remove	R/W	Add	Remove
At/near start						
In middle						
At end						

L = (4, 2, 7, 8, 1)

- first(L)
- rest(L)
- cons(element, L)

Loops, Example 1: Max-area Triangle with Origin

Given an array of n 2D points, find a pair of points that maximizes the area of the triangle formed with the origin.

```
function max_triangle(arr)
    origin = (0, 0)

    max_area = 0
    best_x = null
    best_y = null

    for i from 1 to n
        x = arr[i]
        for j from i + 1 to n
            y = arr[j]

            a = dist(x, origin) // length of side a
            b = dist(y, origin) // length of side b
            c = dist(x, y)      // length of side c

            s = (a + b + c) / 2 // semiperimeter

            area = sqrt(s * (s - a) * (s - b) * (s - c)) // Heron's Formula

            if area > max_area
                max_area = area
                best_x = x
                best_y = y

    return max_area // Could also return best_x and best_y if desired
```

Loops, Example 1: Max-area Triangle with Origin

Given an array of n 2D points, find a pair of points that maximizes the area of the triangle formed with the origin.

```
function max_triangle(arr)
    origin = (0, 0)

    max_area = 0
    best_x = null
    best_y = null

    for i from 1 to n
        x = arr[i]
        for j from i + 1 to n
            y = arr[j]

            a = dist(x, origin) // length of side a
            b = dist(y, origin) // length of side b
            c = dist(x, y)      // length of side c

            s = (a + b + c) / 2 // semiperimeter

            area = sqrt(s * (s - a) * (s - b) * (s - c)) // Heron's Formula

            if area > max_area
                max_area = area
                best_x = x
                best_y = y

    return max_area // Could also return best_x and best_y if desired
```

Loops, Example 2: Graph Reachability

Given a graph G and a start vertex s, find all nodes reachable from s (i.e., in the same connected component as s).

```
function component(G = (V, E), s)
    for v in V
        unmark(v)

    RV = emptylist
    Q = emptylist
    mark(s)
    cons(s, Q)

    while (Q is not empty)
        u = first(Q)
        Q = rest(Q)
        cons(u, RV)

        for every v in V such that uv in E
            if v is not marked
                mark v
                cons(v, Q)

    return RV

```

Loops, Example 2: Graph Reachability

Given a graph G and a start vertex s, find all nodes reachable from s (i.e., in the same connected component as s).

```
function component(G = (V, E), s)
    for v in V
        unmark(v)

    RV = emptylist
    Q = emptylist
    mark(s)
    cons(s, Q)

    while (Q is not empty)
        u = first(Q)
        Q = rest(Q)
        cons(u, RV)

        for every v in V such that uv in E
            if v is not marked
                mark v
                cons(v, Q)

    return RV

```

Learning Objectives

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

- Be able to read code that uses basic linked list operations (first, rest, cons).
- Know the big-O running times of basic operations on linked lists and arrays.
- For an algorithm involving loops (perhaps nested), express its running time using summations.
- Given an unfamiliar but fairly simple function in pseudo-code, analyze how long it takes using big-O notation.