UCLA Samueli Computer Science

CS32: Introduction to Computer Science II

Discussion Week 8

Junheng Hao, Arabelle Siahaan

May 24, 2019

Announcements

- Homework 4 is due 11PM next Tuesday, May 28. (@A@)
- How do you feel about midterm 2?

Outline Today

UCLA Samueli Computer Science

- Sorting (cont'd)
- Tree (Part 1)
- Homework 4 Guidelines

Sorting
Introduction

UCLA Samueli Computer Science

Most important algorithm ever!

Methods:

- Selection sort
- Bubble sort
- Insertion sort
- Merge sort
- Quick sort

Focus on:

1. Steps for each sorting algorithm
2. Runtime complexity for worst cases, best cases and average cases
3. Space complexity
4. How about additional assumptions, such as the array is “almost sorted” / “reversed” arrays

Sorting

Selection sort

Steps:

4	3	1	5	2
1	3	4	5	2
1	2	4	5	3
1	2	3	5	4
1	2	3	4	5

Idea: Find the smallest item in the unsorted portion and place it in the front.

Runtime complexity:

Average: $O(n^2)$

Worst: $O(n^2)$

Best: $O(n^2)$

Space complexity: $O(1)$

Sorting

Insertion sort

Steps:

4	3	1	5	2
3	4	1	5	2
1	3	4	5	2
1	3	4	5	2
1	2	3	4	5

Idea: Pick one from the unsorted part and place it in the right position.

Runtime complexity:

Average: $O(n^2)$

Worst: $O(n^2)$

Best: $O(n)$

Space complexity: $O(1)$

Sorting

Bubble sort

Steps:

4	3	1	5	2
3	4	1	5	2
3	1	4	5	2
3	1	4	2	5
1	3	2	4	5
1	2	3	4	5

Idea: Well, just “bubble” as its name

Runtime complexity:

Average: $O(n^2)$

Worst: $O(n^2)$

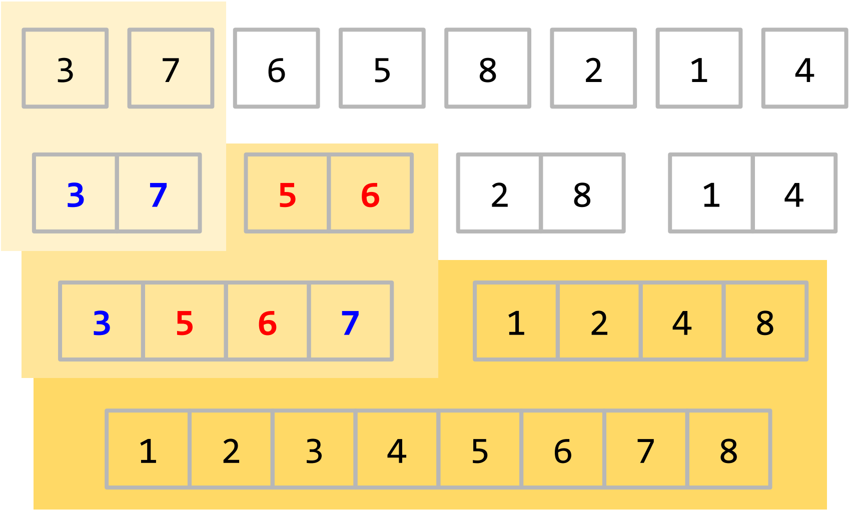
Best: $O(n)$

Space complexity: $O(1)$

Sorting

Merge sort

Steps:



Idea: Divide and conquer

Runtime complexity:

Average: $O(n \log n)$

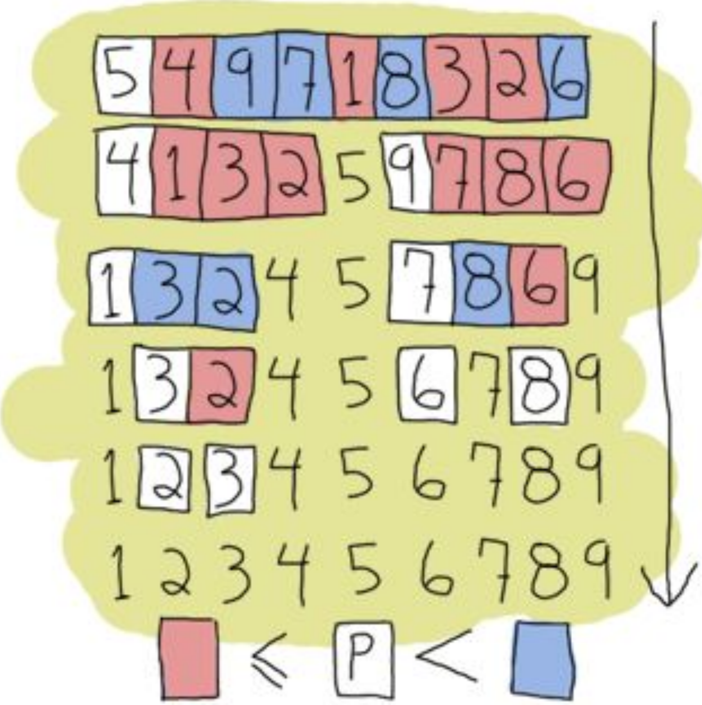
Worst: $O(n \log n)$

Best: $O(n \log n)$

Space complexity: $O(n)$

Sorting

Quicksort



Idea: Set a pivot. Numbers less then pivot are placed to front while other to end.

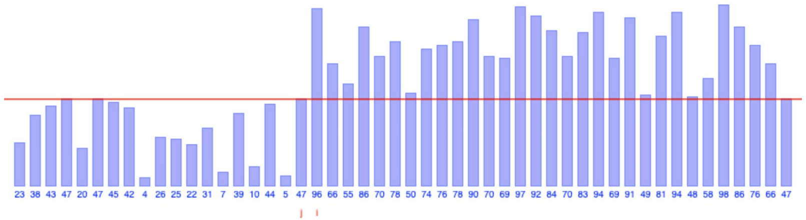
Runtime complexity:

Average: $O(n \log n)$

Worst: $O(n^2)$

Best: $O(n \log n)$

Space complexity: $O(\log n)$

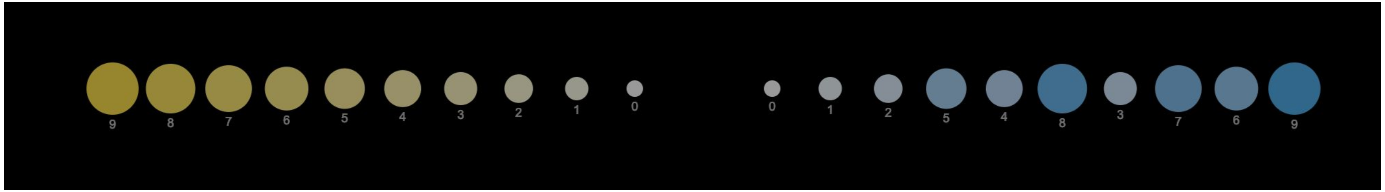


Sorting

Other methods and complexity?

- O(n log n) is faster than O(n^2) → Merge sort is more efficient than selection, insertion and bubble sort in runtime.
- O(n log n) is best average complexity that a general sorting algorithm can achieve.
- With more information about the data provided, you can sometimes sort things almost linearly.

Question: What is the complexity of these sorting algorithms if you know the array is **reversed**? What if the array is **almost already sorted**?

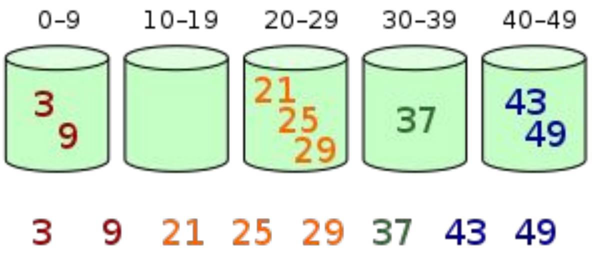


Sorting

Other methods and complexity?

There are many other sorting methods:

- Shell sort (shell 1959, Knuth 1973, Ciura 2001)
- Quicksort 3-way
- Heap sort
- Bucket sort



Sorting

Why sorting is important?

Sorting is the most important and basic algorithm. Many other real-world problems are somewhat based on sorting, including:

Sorting Algorithms Animations: https://www.toptal.com/developers/sorting-algorithms
Other good demos:
https://www.cs.usfca.edu/~galles/visualization/ComparisonSort.html
http://sorting.at/

Sorting

Variant sorting problems - Example

UCLA Samueli Computer Science

Question: How about get the *K-th* largest numbers in one array?

Leetcode question #215

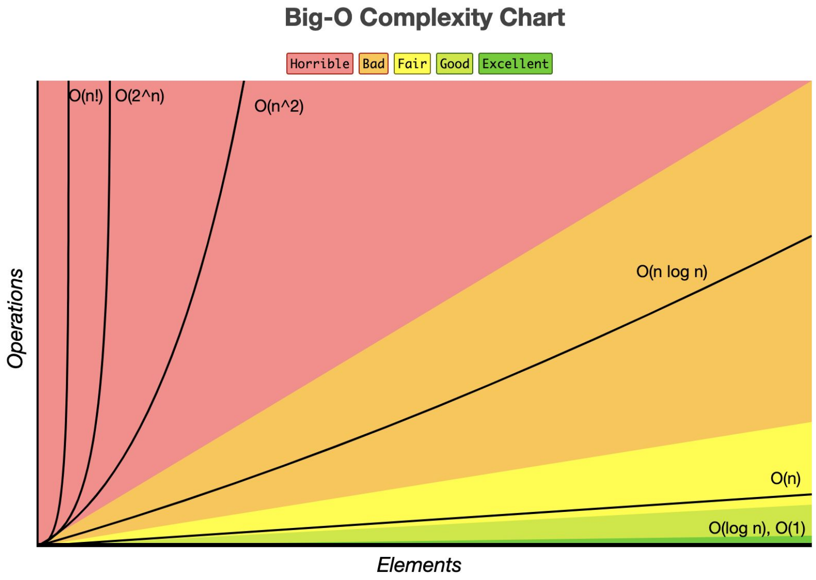
Hint:

1. How to find the k-th largest numbers by merge sort and quicksort (or other sort methods)? What are the average and worst complexity?
2. What data structures is good to use?

Big-O Notation

Big-O Complexity Chart

UCLA Samueli Computer Science



Array Sorting Algorithms

Algorithm	Time Complexity			Space Complexity
	Best	Average	Worst	Worst
Quicksort	Ω(n log(n))	Θ(n log(n))	O(n^2)	O(log(n))
Mergesort	Ω(n log(n))	Θ(n log(n))	O(n log(n))	O(n)
Timsort	Ω(n)	Θ(n log(n))	O(n log(n))	O(n)
Heapsort	Ω(n log(n))	Θ(n log(n))	O(n log(n))	O(1)
Bubble Sort	Ω(n)	Θ(n^2)	O(n^2)	O(1)
Insertion Sort	Ω(n)	Θ(n^2)	O(n^2)	O(1)
Selection Sort	Ω(n^2)	Θ(n^2)	O(n^2)	O(1)
Tree Sort	Ω(n log(n))	Θ(n log(n))	O(n^2)	O(n)
Shell Sort	Ω(n log(n))	Θ(n(log(n))^2)	O(n(log(n))^2)	O(1)
Bucket Sort	Ω(n+k)	Θ(n+k)	O(n^2)	O(n)
Radix Sort	Ω(nk)	Θ(nk)	O(nk)	O(n+k)
Counting Sort	Ω(n+k)	Θ(n+k)	O(n+k)	O(k)
Cubesort	Ω(n)	Θ(n log(n))	O(n log(n))	O(n)

Tree

Definition

- Terms: Node/edge, root node, leaf node, parent and child node, subtree, levels (height/depth).
- Features: No loop, no shared children
- Question: How many edges should there be in a tree with ***n*** nodes?
- Binary tree: no node has more than two children.
- Question: How many nodes can a binary tree of height ***h*** have? → Full binary tree

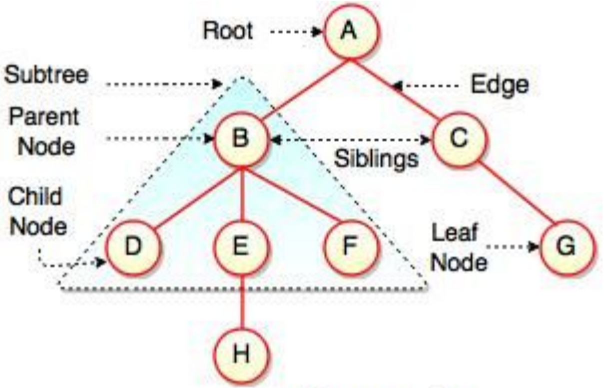
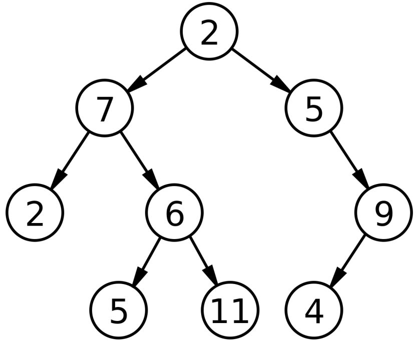


Fig. Structure of Tree



Tree

As a data structure

UCLA Samueli Computer Science

- Tree is a useful data structure!
- Basic functions: insert, remove, search, **traverse**
- How to traverse a tree?

```
struct Node{
  ItemType val;
  Node* leftChild;
  Node* rightChild;
} // a simple node
```

```
Class Tree{
public:
  // ???
private:
  // ???
}
```

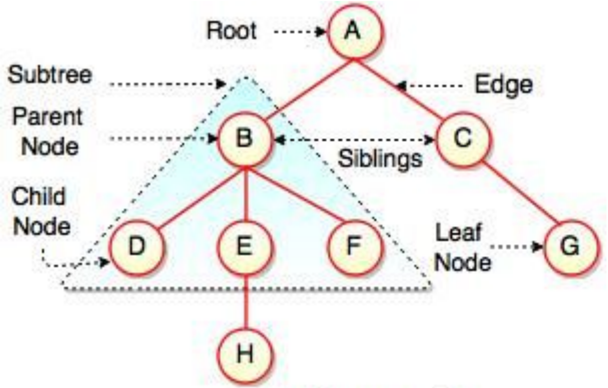
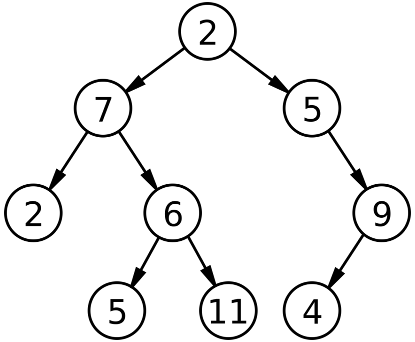


Fig. Structure of Tree



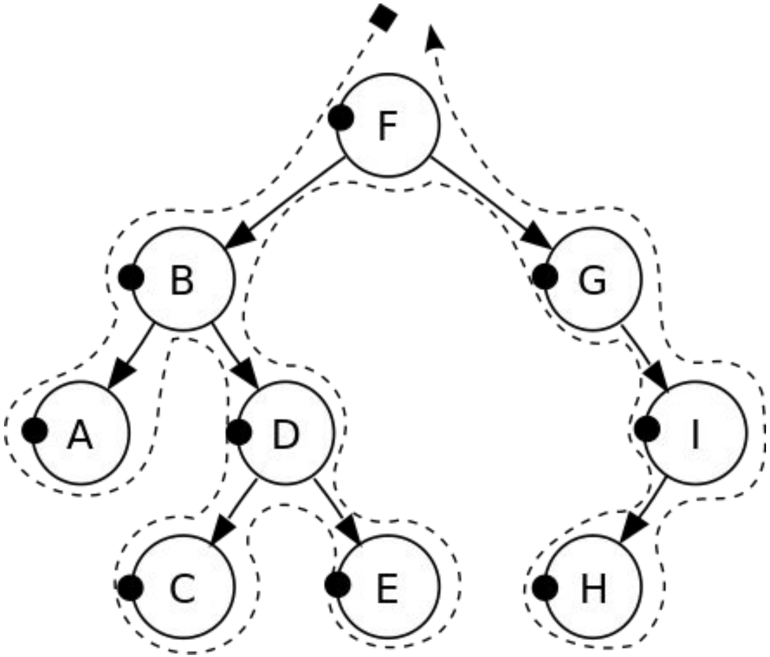
Tree Traversal: Pre-order

Three methods of tree traversal


```
void preorder(const Node* node)
{
  if (node == nullptr) return;
  cout << node->val << “,”;
  preorder(node->left);
  preorder(node->right);
}
```

Pre-order output:

`F, B, A, D, C, E, G, I, H`



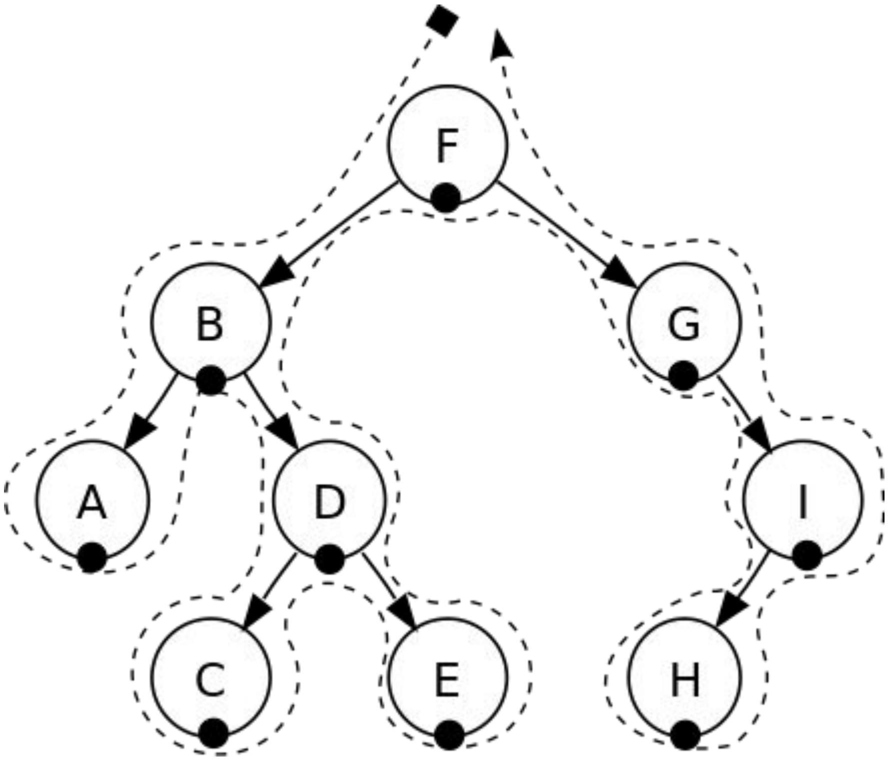
Tree Traversal: In-order

Three methods of tree traversal

```
void inorder(const Node* node)
{
  if (node == nullptr) return;
  inorder(node->left);
  cout << node->val << “,”;
  inorder(node->right);
}
```

In-order output:

`A, B, C, D, E, F, G, H, I`



Tree Traversal: Post-order

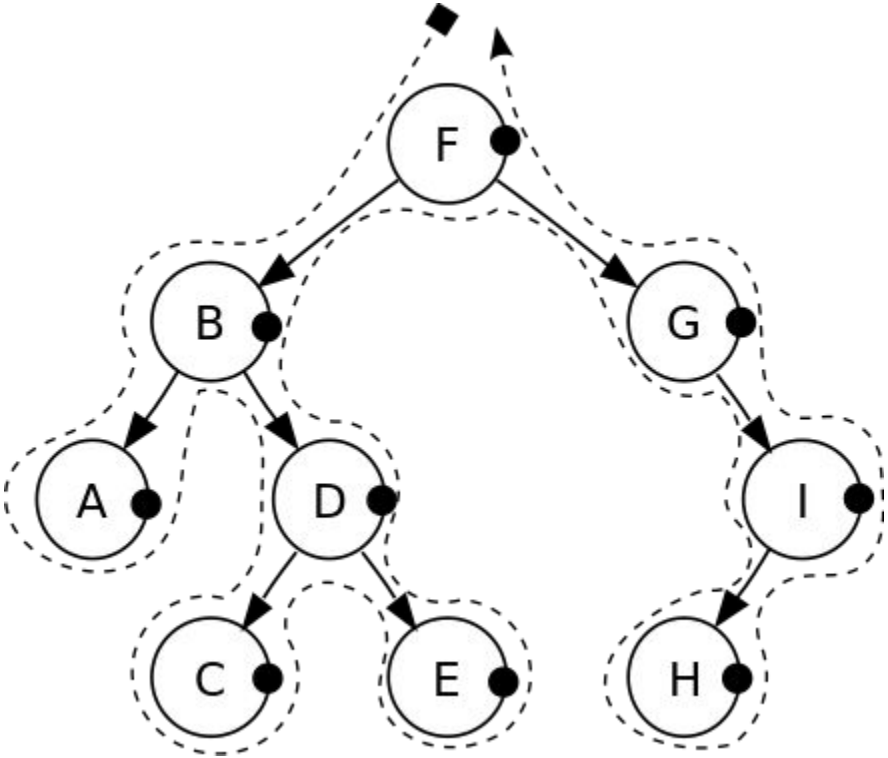
Three methods of tree traversal

UCLA Samueli Computer Science

```
void postorder(const Node* node)
{
  if (node == nullptr) return;
  postorder(node->left);
  postorder(node->right);
  cout << node->val << “,”;
}
```

Post-order output:

`A, C, E, D, B, H, I, G, F`



Tree Traversal: Compare

UCLA Samueli Computer Science

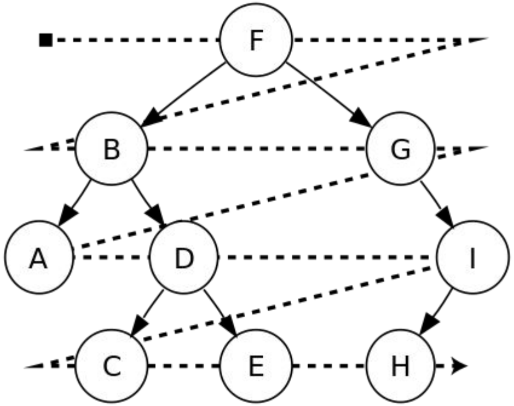
```
void preorder(const Node* node)
{
  if (node == nullptr) return;
  cout << node->val << ",";
  preorder(node->left);
  preorder(node->right);
}
```

```
void inorder(const Node* node)
{
  if (node == nullptr) return;
  inorder(node->left);
  cout << node->val << ",";
  inorder(node->right);
}
```

```
void postorder(const Node* node)
{
  if (node == nullptr) return;
  postorder(node->left);
  postorder(node->right);
  cout << node->val << ",";
}
```

```
//Other ways?
```

Level-order or say breadth-first search!

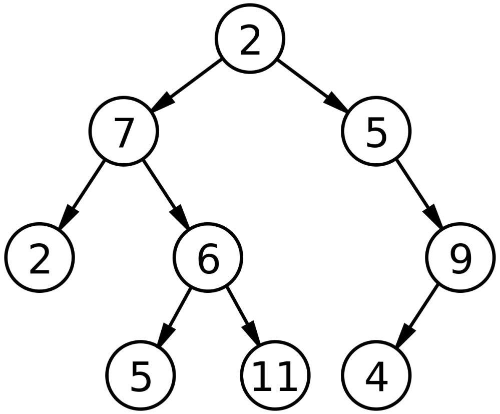


Tree

Recursion in trees

- It is easy and natural to apply recursion on trees!
- Pre-order / in-order / post-order are all recursive methods to traverse a tree.
- Question: How to calculate a height of a given tree?

```
int getTreeHeight(const Node* node)
{
  if (node == nullptr) return 0;
  int leftHeight = getTreeHeight(node->left);
  int rightHeight = getTreeHeight(node->right);
  if (leftHeight > rightHeight)
    return leftHeight + 1;
  else
    return rightHeight + 1;
}
```



Tree

Conclusions and applications

- Non-linear data structures unlike arrays or lists.
- Present hierarchy
- Optimized (like balanced) tree and variants can improve efficiency of search, sort and other typical problems.
- Other tree variants:
  - B tree / B+ tree
  - Red-black tree
  - K-D Tree
  - Suffix Tree

```
file system
-----------
     /    <-- root
  /     \
...    home
      /    \
   ugrad    course
    /      /   |   \
  ...   cs31 cs32 cs35L
```

Next week about tree

- Binary Search Tree
- Heap
- More interesting questions …

6 5 3 1 8 7 2 4

Hints for Homework 4 - P1

UCLA Samueli Computer Science

```
template <typename T>
class S
{
  ...
  struct N
  {
    ...
  };
  N* f();
  ...
};

template <typename T>
typename S<T>::N* S<T>::f()       // OK
{
  ...
}
```

Note & Reminders:

1. Only need to change (or add) a few lines if you decide to use template **`<typename ItemType>`**
2. Put **`template <typename T>`** ahead of every function you declare outside the class statement.

Hints for Homework 4 - P2

UCLA Samueli Computer Science

```
#include "Sequence.h"  // class template from problem 1
      class Coord
      {
        public:
          Coord(int r, int c) : m_r(r), m_c(c) {}
          Coord() : m_r(0), m_c(0) {}
          double r() const { return m_r; }
          double c() const { return m_c; }
        private:
          double m_r;
          double m_c;
      };
      int main()
      {
          Sequence<int> si;
          si.insert(50);                // OK
          Sequence<Coord> sc;
          sc.insert(0, Coord(50,20));   // OK
          sc.insert(Coord(40,10));      // error!
      }
```

Note & Reminders:

1. Check the 1-argument `insert()` function and think about the operation which leads to error for Coord but not for `int`.

Hints for Homework 4 - P3, P4

UCLA Samueli Computer Science

```
// Problem 4

void listAll(const Domain* d, string path)
// two-parameter overload
{
    // You will write this code.
}

void listAll(const Domain* d)
// one-parameter overload
{
    if (d != nullptr)
        listAll(d, d->label());
}
```

Note & Reminders:

1. (P3) Implement functions with `remove()` in STL.
2. (P4) Necessity for two-parameter overload of `listAll()` function. → What is the purpose of `string path` parameter?
3. (P4) If the container is const, you must use a `const_iterator` to traverse an STL container.

Hints for Homework 4 - P5, 6, 7

UCLA Samueli Computer Science

Note & Reminders:

1. Only the highest order will be considered as your final answer.
2. Analysis implementation of **every** function in **every** step.

Break Time! (5 minutes)

Q & A

Group Exercises: Worksheet

- Exercise problems from **Worksheet 7** (see “LA worksheet” tab in CS32 website). Answers will be posted next week.

- Questions for today:
    - TBA

Question X

UCLA Samueli Computer Science

Content

UCLA Samueli Computer Science

Thank you!

Q & A