Recap on STL:

- vector

Announcements:

- Homework #1 is due, late submission will be better than no submission
- For 815 project, rank your preference by Friday 11:59pm
- Utilize office hours (9:30AM-12:30PM Friday) for further questions
- When classroom is full, the seating priority should be given to enrolled students
Recap on STL: string and map

```cpp
#include <string>
#include <map>

int main(int argc, char** argv) {
    // string examples
    std::string s("hello");
    std::cout << s << std::endl; // prints "hello"
    s += "o;"; // use ']' for concatenation
    std::cout << s << std::endl; // prints "hello"
    s[0] = 'j'; // access & assign individual character via []
    std::cout << s << std::endl; // prints "jello"
    std::sort(s.begin(), s.end()); // string is a vector of characters
    std::cout << s << std::endl; // prints "ejillo"
    // map examples
    std::map<std::string, int> points;
    points["Carlo Sidore"] = 100;
    points["Goncalo Abecasis"] = 99;
    points["Hyun Min Kang"] = 50;
    std::cout << points["Carlo Sidore"] << std::endl; // prints 100
    std::cout << points["Goncalo Abecasis"] << std::endl; // prints 99
    std::cout << points["Hyun Min Kang"] << std::endl; // nonexistent key; prints default value
}
```

Reading from Files: stdSort.cpp

```cpp
#include <iostream>

int main(int argc, char** argv) {
    // stdSort.cpp
    std::ifstream fin(argv[1]);
    if (argc > 1) { // if argument is given, read from file
        std::ifstream fin(argv[1]);
        while (fin >> tok) { v.push_back(tok); }
        fin.close();
    } else { // read from standard input if no argument is specified
        while (std::cin >> tok) { v.push_back(tok); }
    }
    std::sort(v.begin(), v.end()); // sort using the algorithm in STL
    for (int i=0; i < v.size(); ++i) {
        std::cout << v[i] << std::endl; // print out the content
    }
    return 0;
}
```

Reading from Files: insertionSort.cpp

```cpp
#include <iostream>

int main(int argc, char** argv) {
    // insertionSort.cpp
    std::ifstream fin(argv[1]);
    if (argc > 1) { // if argument is given, read from file
        std::ifstream fin(argv[1]);
        while (fin >> tok) { v.push_back(tok); }
        fin.close();
    } else { // read from standard input if no argument is specified
        while (std::cin >> tok) { v.push_back(tok); }
    }
    insertionSort(v); // differs from stdSort in only this part
    for (int i=0; i < v.size(); ++i) {
        std::cout << v[i] << std::endl;
    }
    return 0;
}
```

Running time comparison

```cpp
#include <iostream>
#include <fstream>
#include <vector>

void insertionSort(std::vector<int>& v) { // insertionSort as defined before
    int tok;
    std::vector<int> v;
    if (argc > 1) {
        std::ifstream fin(argv[1]);
        while (fin >> tok) { v.push_back(tok); }
        fin.close();
    } else {
        while (std::cin >> tok) { v.push_back(tok); }
    }
    insertionSort(v); // differs from stdSort in only this part
    for (int i=0; i < v.size(); ++i) {
        std::cout << v[i] << std::endl;
    }
    return 0;
}
```

Running example with 100,000 elements (in UNIX or MacOS)

```bash
time cat src/sample.input.txt | src/stdSort > /dev/null
real 0m0.430s
user 0m0.281s
sys 0m0.130s
```

```bash
time cat src/sample.input.txt | src/insertionSort > /dev/null
real 1m8.795s
user 1m8.181s
sys 0m0.206s
```
Recursion

Definition of recursion
Recursion See "Recursion".

Another definition of recursion
Recursion If you still don’t get it, see: “Recursion”

Key components of recursion
• A function that is part of its own definition
• Terminating condition (to avoid infinite recursion)

Euclid’s algorithm

Algorithm GCD
Data: Two integers $a$ and $b$
Result: The greatest common divisor (GCD) between $a$ and $b$

if $a$ divides $b$ then
  return $a$
else
  Find the largest integer $t$ such that $at + r = b$
  return GCD($r$, $a$)
end

Function gcd()
int gcd (int $a$, int $b$) {
  if ( $a$ == 0 ) return $b$; // equivalent to returning $a$ when $b$ % $a$ == 0
  else return gcd( $b$ % $a$, $a$ );
}

Example of recursion

Factorial
int factorial(int $n$) {
  if ( $n$ == 0 )
    return 1;
  else
    return $n$ * factorial($n$-1); // tail recursion - can be transformed into loop
}

towerOfHanoi
void towerOfHanoi(int $n$, int $s$, int $i$, int $d$) { // $n$ disks, from $s$ to $d$ via $i$
  if ( $n$ > 0 ) {
    towerOfHanoi($n$-1,$s$,$d$,$i$); // recursively move $n$-1 disks from $s$ to $d$
    std::cout << "Disk " << $n$ << ": " << $s$ << " -> " << $d$ << std::endl;
    towerOfHanoi($n$-1,$i$,$s$,$d$); // recursively move $n$-1 disks from $i$ to $d$
  }
}

A running example of Euclid’s algorithm

Function gcd()
int gcd (int $a$, int $b$) {
  if ( $a$ == 0 ) return $b$; // equivalent to returning $a$ when $b$ % $a$ == 0
  else return gcd( $b$ % $a$, $a$ );
}

Evaluation of gcd(477, 246)
gcd(477, 246)
gcd(231, 246)
gcd(15, 231)
gcd(6, 15)
gcd(3, 6)
gcd(0, 3)
gcd(477, 246) == 3
Divide-and-conquer algorithms

Solve a problem recursively, applying three steps at each level of recursion:

- **Divide**: the problem into a number of subproblems that are smaller instances of the same problem.
- **Conquer**: the subproblems by solving them recursively. If the subproblem sizes are small enough, however, just solve the subproblems in a straightforward manner.
- **Combine**: the solutions to subproblems into the solution for the original problem.

Binary Search

// assuming a is sorted, return index of array containing the key, // among a[start...end]. Return -1 if no key is found
t

t

t

Recursive Maximum

// find maximum within an a[start...end]
int findMax(std::vector<int>& a, int start, int end) {
  if ( start == end ) return a[start]; // conquer small problem directly
  else {
    int mid = (start+end)/2;
    int leftMax = findMax(a,start,mid); // divide the problem into half
    int rightMax = findMax(a,mid+1,end);
    return ( leftMax > rightMax ? leftMax : rightMax ); // combine solutions
  }
}

Merge Sort

Divide and conquer algorithm

- **Divide**: Divide the n element sequence to be sorted into two subsequences of n/2 elements each
- **Conquer**: Sort the two subsequences recursively using merge sort
- **Combine**: Merge the two sorted subsequences to produce the sorted answer

http://www.sorting-algorithms.com/merge-sort
mergeSort.cpp - main()

```cpp
#include <iostream>
#include <vector>
#include <limits>

void mergeSort(std::vector<int>& a, int p, int r);

int main(int argc, char** argv) {
    int tok;
    std::vector<int> v;
    if (argc > 1) {
        std::ifstream fin(argv[1]);
        while (fin >> tok) { v.push_back(tok); }
    }
    fin.close();
    else {
        while (std::cin >> tok) { v.push_back(tok); }
    }
    mergeSort(v, 0, v.size() - 1); // same as before except for this line
    for (int i = 0; i < v.size(); ++i) {
        std::cout << v[i] << std::endl;
    }
    return 0;
}
```

mergeSort.cpp - merge() function

```cpp
void merge(std::vector<int>& a, int p, int q, int r) {
    std::vector<int> aL, aR;
    // copy a[p..q-1] to aL and a[q..r] to aR
    for (int i = p; i < q; ++i) aL.push_back(a[i]);
    for (int i = q; i <= r; ++i) aR.push_back(a[i]);
    aL.push_back(INT_MAX); // append additional value to avoid out-of-bound
    aR.push_back(INT_MAX);
    // scan sorted aL and aR separately, taking the minimum between the two
    for (int k = p, i = 0, j = 0; k <= r; ++k) {
        if (aL[i] < aR[j]) { a[k] = aL[i]; ++i; }
        else { a[k] = aR[j]; ++j; }
    }
}
```

Time Complexity of Merge Sort

If \( n = 2^m \)

```latex
T(n) = \begin{cases} 
    c & \text{if } n = 1 \\
    2T(n/2) + cn & \text{if } n > 1 
\end{cases}
```

\[
T(n) = \sum_{i=1}^{m} cn = cnm = cn \log_2(n) = \Theta(n \log_2 n)
\]

For arbitrary \( n \)

\[
T(n) = \begin{cases} 
    c & \text{if } n = 1 \\
    T([n/2]) + T([n/2]) + cn & \text{if } n > 1 
\end{cases}
\]

\[
cn\log_2 n \leq T(n) \leq cn\log_2 n
\]

\[
T(n) = \Theta(n \log_2 n)
\]
Running time comparison

Running example with 100,000 elements (in UNIX or MacOS)

```plaintext
user@host:~/> time cat src/sample.input.txt | src/stdSort > /dev/null
real 0m0.430s
user 0m0.281s
sys 0m0.130s

user@host:~/> time cat src/sample.input.txt | src/insertionSort > /dev/null
real 1m8.181s
user 1m8.795s
sys 0m0.281s

user@host:~/> time cat src/sample.input.txt | src/mergeSort > /dev/null
real 0m0.898s
user 0m0.755s
sys 0m0.131s
```

Introduction

STL

Recursion

Gcd

Divide and Conquer

MergeSort

QuickSort

QuickSort Algorithm

Algorithm QUICKSORT

Data: array A and indices p and r
Result: A[p..r] is sorted

if p < r then
    q = PARTITION(A,p,r);
    QUICKSORT(A,p,q-1);
    QUICKSORT(A,q+1,r);
end

QuickSort Overview

- Worse-case time complexity is \( \Theta(n^2) \)
- Expected running time is \( \Theta(n \log_2 n) \).
- But in practice mostly performs the best

Divide and conquer algorithm

**Divide** Partition (rearrange) the array \( A[p..r] \) into two subarrays
- Each element of \( A[p..q-1] \leq A[q] \)
- Each element of \( A[q+1..r] \geq A[q] \)

Compute the index \( q \) as part of this partitioning procedure

**Conquer** Sort the two subarrays by recursively calling quicksort

**Combine** Because the subarrays are already sorted, no work is needed to combine them. The entire array \( A[p..r] \) is now sorted

QuickSort Algorithm

Algorithm PARTITION

Data: array \( A \) and indices \( p \) and \( r \)
Result: Returns \( q \) such that \( A[p..q-1] \leq A[q] \leq A[q+1..r] \)

\( x = A[r] \);
\( i = p - 1 \);

for \( j = p \) to \( r - 1 \) do
    if \( A[j] \leq x \) then
        if \( i = j \) then
            \( i = i + 1 \);
        \( \text{EXCHANGE}(A[i],A[j]) \);
    end
end

\( \text{EXCHANGE}(A[i+1],A[r]) \);
return \( i + 1 \);
How Partition Algorithm Works

![Partition Algorithm Diagram]

Implementation of QUICKSORT Algorithm

```cpp
// quickSort function
// The main function is the same to mergeSort.cpp except for the function name
void quickSort(std::vector<int>& A, int p, int r) {
    if ( p < r ) { // immediately terminate if subarray size is 1
        int piv = A[r]; // take a pivot value
        int i = p-1; // p-1 is the # elements < piv among A[p..j]
        int tmp;
        for(int j=p; j < r; ++j) {
            if ( A[j] < piv ) { // if smaller value is found, increase q (=i+1)
                ++i;
            }
        }
        quickSort(A, p, i);
        quickSort(A, i+2, r);
    }
}
```

Running time comparison

Running example with 100,000 elements (in UNIX or MacOS)

```bash
user@host:~/> time cat src/sample.input.txt | src/stdSort > /dev/null
real 0m0.430s
user 0m0.281s
sys 0m0.130s

user@host:~/> time cat src/sample.input.txt | src/insertionSort > /dev/null
real 1m8.795s
user 1m8.181s
sys 0m0.206s

user@host:~/> time cat src/sample.input.txt | src/mergeSort > /dev/null
real 0m0.898s
user 0m0.755s
sys 0m0.131s

user@host:~/> time cat src/sample.input.txt | src/quickSort > /dev/null
real 0m0.427s
user 0m0.285s
sys 0m0.129s
```

Reading Material

- CLRS Chapter 2
- CLRS Chapter 3
- CLRS Chapter 4
- CLRS Chapter 7