VE281 RC 1

Asymptotic Analysis

Big-O

$T(n) \leq cf(n)$ for all $n > n_0$. then $T(n) = O(f(n))$ (actually, $T(n) \in O(f(n))$ since $O(f(n))$ is a set of functions)

Example 1:

$3n = O(2n)$

$3^n = O(2^n)$

$n^3 = O(n^2)$

$\log_2 n = O(\log_3 n)$

Big-Omega

$T(n) \geq cf(n)$ for all $n > n_0$. then $T(n) = \Omega(f(n))$

Big-Theta

Both big-O and big-Omega.

Can you understand all the examples in the slides?

Example 2

```
sum = 0;
for(i = 1; i <= n; i *= 2)
  for(j = 1; j <= i; j++)
    sum++;
```

A. $\Theta(\log n)$ B. $\Theta(n \log n)$
C. $\Theta(n)$ D. $\Theta(n^2)$

Master Method

$T(n) \leq aT(\frac{n}{b}) + O(n^d)$

- if $a = b^d$, $T(n) = O(n^d \log n)$
- if $a < b^d$, $T(n) = O(n^d)$
- if $a > b^d$, $T(n) = O(n^{log_b a})$

The essence here is to compare the recursive term and the regular term, determine who is the determinist term.

Comparison Sort (Very Important)

Insertion Sort

- **A[0] alone is a sorted array.**
- **For i=1 to N-1 – Insert A[i] into the appropriate location in the sorted array A[0], …, A[i-1], so that A[0], …, A[i] is sorted. – To do so, save A[i] in a temporary variable t, shift sorted elements greater than t right, and then insert t in the gap.**

Selection Sort

- **For i=0 to N-2 – Find the smallest item in the array A[i], …, A[N-1]. Then, swap that item with A[i].**

Bubble Sort

- **For i=N-2 downto 0**
  - **For j=0 to i**
    - **If A[j]>A[j+1] swap A[j] and A[j+1]**

Merge Sort

```
void mergesort(int *a, int left, int
  right) {
    if (left >= right) return;
    int mid = (left+right)/2;
    mergesort(a, left, mid);
    mergesort(a, mid+1, right);
    merge(a, left, mid, right);
}
```

The key of merge sort is "merge".

- **Compare the smallest element in the two arrays A and B and move the smaller one to an additional array C.**
- **Repeat until one of the arrays becomes empty.**

- **Then append the other array at the end of array C.**

Quick Sort

- **Choose an array element as pivot.**
- **Put all elements < pivot to the left of pivot.**
- **Put all elements ≥ pivot to the right of pivot.**
- **Move pivot to its correct place in the array.**
- **Sort left and right subarrays recursively (not including pivot).**

The choice of pivot is important. If the pivot is not around the median, the partitioned array is not balanced.

Example 3 Case study

1, **3**, 8, 5, 6, 9, 4, *3*, 7, 2

Insertion Sort

Selection Sort

Bubble Sort

Merge Sort

Quick Sort

Example 4 (If you cannot understand it, remember it.)

	Worst Time	Average Time	Whether in place?	Whether stable?
Insertion				
Selection				
Bubble				
Merge Sort				
Quick Sort				

What is the best sorting strategy? What algorithm is the garbage?

Code Detail

Insertion Sort

```cpp
#include <iostream>

// Function to perform Insertion Sort
void insertionSort(int arr[], int n) {
}

// Function to print an array
void printArray(int arr[], int n) {
    for (int i = 0; i < n; i++) {
        std::cout << arr[i] << " ";
    }
    std::cout << std::endl;
}

int main() {
    int arr[] = {12, 11, 13, 5, 6};
    int n = sizeof(arr) / sizeof(arr[0]);

    std::cout << "Original Array: ";
    printArray(arr, n);

    insertionSort(arr, n);

    std::cout << "Sorted Array: ";
    printArray(arr, n);

    return 0;
}
```

Selection Sort

```cpp
#include <iostream>

// Function to perform Selection Sort
void selectionSort(int arr[], int n) {
    for (int i = 0; i < n - 1; i++) {
        int minIndex = i;
        for (int j = i + 1; j < n; j++) {
            if (arr[j] < arr[minIndex]) {
                minIndex = j;
            }
        }
        // Swap the found minimum element with the current element
        int temp = arr[i];
        arr[i] = arr[minIndex];
        arr[minIndex] = temp;
    }
}

// Function to print an array
void printArray(int arr[], int n) {
```

```
    for (int i = 0; i < n; i++) {
        std::cout << arr[i] << " ";
    }
    std::cout << std::endl;
}

int main() {
    int arr[] = {64, 25, 12, 22, 11};
    int n = sizeof(arr) / sizeof(arr[0]);

    std::cout << "Original Array: ";
    printArray(arr, n);

    selectionSort(arr, n);

    std::cout << "Sorted Array: ";
    printArray(arr, n);

    return 0;
}
```

Bubble Sort

```
#include <iostream>

// Function to perform Bubble Sort
void bubbleSort(int arr[], int n) {
}

// Function to print an array
void printArray(int arr[], int n) {
    for (int i = 0; i < n; i++) {
        std::cout << arr[i] << " ";
    }
    std::cout << std::endl;
}

int main() {
    int arr[] = {64, 34, 25, 12, 22, 11, 90};
    int n = sizeof(arr) / sizeof(arr[0]);

    std::cout << "Original Array: ";
    printArray(arr, n);

    bubbleSort(arr, n);

    std::cout << "Sorted Array: ";
    printArray(arr, n);

    return 0;
}
```

Merge Sort

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

// Merge two subarrays of arr[]
// First subarray is arr[l..m]
// Second subarray is arr[m+1..r]
void merge(std::vector<int>& arr, int l, int m, int r) {
    int n1 = m - l + 1;
    int n2 = r - m;

    // Create temporary arrays
    std::vector<int> L(n1);
    std::vector<int> R(n2);

    // Put your code here

}

// Main function to perform Merge Sort
void mergeSort(std::vector<int>& arr, int l, int r) {
    if (l < r) {
        // Same as (l+r)/2, but avoids overflow for large l and r
        int m = l + (r - l) / 2;

        // Sort first and second halves
        // Put your code here

        // Merge the sorted halves
        merge(arr, l, m, r);
    }
}

// Function to print an array
void printArray(const std::vector<int>& arr) {
    for (int num : arr) {
        std::cout << num << " ";
    }
    std::cout << std::endl;
}

int main() {
    std::vector<int> arr = {12, 11, 13, 5, 6, 7};
    int n = arr.size();

    std::cout << "Original Array: ";
    printArray(arr);

    mergeSort(arr, 0, n - 1);

    std::cout << "Sorted Array: ";
    printArray(arr);

    return 0;
```

```
}
```

Quick Sort: The implementation is a little bit different from the pseudo code in the slides. Both are OK. Won't appear in the exam.

```
#include <iostream>
#include <vector>

// Function to partition the array into two subarrays based on a pivot element
// Elements smaller than the pivot are on the left, and elements greater than
// the pivot are on the right.
int partition(std::vector<int>& arr, int low, int high) {
    int pivot = arr[high]; // Choose the rightmost element as the pivot
    int i = (low - 1); // Index of the smaller element

    //Put your code here
}

// Function to perform Quick Sort
void quickSort(std::vector<int>& arr, int low, int high) {
    if (low < high) {
        // Partition the array into two subarrays
        int pi = partition(arr, low, high);

        // Recursively sort the subarrays
        // Put your code here
    }
}

// Function to print an array
void printArray(const std::vector<int>& arr) {
    for (int num : arr) {
        std::cout << num << " ";
    }
    std::cout << std::endl;
}

int main() {
    std::vector<int> arr = {10, 7, 8, 9, 1, 5};
    int n = arr.size();

    std::cout << "Original Array: ";
    printArray(arr);

    quickSort(arr, 0, n - 1);

    std::cout << "Sorted Array: ";
    printArray(arr);

    return 0;
}
```