CSE 3010 – Data Structures & Algorithms

Lecture #28

What will be covered today

- Merge sort algorithm
- Classification of sorting techniques
- Introduction to search techniques

Merge sort algorithm

- Go through the program for Merge sort given in:
  https://www.thecrazyprogrammer.com/2014/03/c-program-for-implementation-of-merge-sort.html

Classification of sorting with examples techniques

Classification	Meaning	Examples
In-place	When no additional space is needed to manipulate the input list while sorting, and if required will implicitly create sub lists within the same structure	1. Bubble sort 2. Selection sort 3. Insertion sort 4. Heap sort 5. Shell sort
Not-in-place	When additional space is required to manipulate the input list while sorting	1. Merge sort

Classification of sorting with examples techniques

Classification	Meaning	Examples
Stable	When two elements having equal values appear in the same order in the sorted list as they appeared in the unsorted list	1. Bubble sort 2. Insertion sort 3. Merge sort 4. Binary tree sort
Unstable	When two elements having equal values do not appear in the same order in the sorted list as they appeared in the unsorted list	1. Selection sort 2. Quick sort 3. Heap sort

Classification of sorting with examples techniques

Classification	Meaning	Examples
Comparison-based	When comparison between elements are done to sort the order of elements in the list	1. Bubble sort 2. Insertion sort 3. Selection sort 4. Quick sort 5. Merge sort
Non-comparison based	When no comparison is performed but use integer arithmetic of the elements	1. Count sort 2. Radix sort

Introduction to search techniques

- Searching for a 'key' in a input dataset
- Types of search techniques
  - Linear search
  - Binary search
  - Jump search
  - Fibonacci search
  - Interpolation search
  - Hashing
  - …

Binary search algorithm

```
bool binarySearch(int numbers[], int numb) {
        int low, high, mid;
        bool found = false;
        low = 0;
        high = SIZE;
        while ((!found) && (low <= high)) {
                mid = (low + high) / 2;
                if (numbers[mid] == numb)
                        found = true;
                else if (numbers[mid] < numb)
                                low = mid + 1;
                        else
                                high = mid - 1;
        }
        if (found)
                return true;
        else
                return false;
}
```