Lab: Sorting and Hashing

Objective

- To practice writing basic sorting algorithms

Reviews

Basic algirithms learned in class

- Insertion Sort
- Selection Sort
- Merge Sort
- Quick Sort

Insertion Sort

- An array is partitioned into sorted and unsorted groups
- At first, the sorted group contains only the first key and all other keys belong to the unsort group
- Each key in the unsorted group is compared with keys in the sorted one and is inserted at an approximate location in the sorted group
- The process is repeated until the unsorted group is empty

5	7	6	3	1
5	7	6	3	1
5	7	6	3	1
5	6	7	3	1
5	6	3	7	1
5	3	6	7	1
3	5	6	7	1
3	5	6	1	7
3	5	1	6	7
3	1	5	6	7
1	3	5	6	7

Selection Sort

- An array is partitioned into sorted and unsorted groups
- At the beginning, the unsorted group contains all keys
- At first round it finds the smallest key from the unsorted group and swaps the smallest with the key at the first location.
- The next round, it finds the smallest key from the remaining unsorted group and swaps the smallest with the key at the next location. Repeat this process until the unsorted group is empty.

5	**7**	**6**	**3**	**1**
1	7	6	3	5
1	3	6	7	5
1	3	5	7	6
1	3	5	6	7

Merge Sort

- Divide: First divide the list into the group of 1 element
- Conquer: compare each element with the adjacent list to sort and merge the two adjacent lists.
- Repeat the divide and conquer processes for the list sizes 2, 4, 8, …, $\log_2 n$

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

Quick Sort

- Divide (partitioning): Select a specific element from the list, which is called *pivot*. Classify data into three sequences: Data less than pivot, equal to pivot, and Data greater than pivot. Recursively sort data less than pivot and also data greater than pivot.
- Conquer: put back the elements in order by first inserting data less than pivot, equal to pivot, and data greater than pivot

Ex. Quick sort with first value used as a pivot

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

.

void swap (int [] A, int i, int j)	swap values of A at indices i and j
void printArray(int [] A)	print the array A with spaces between elements
int findIndexSmallest(int [] A, int start, int end)	find the index of the smallest from indices start to end
int[] mergeSortedArray(int[] A, int[] B)	merge the two sorted arrays A and B and output the new array

Hashing

- Hashing is a technique used for performing insertions, deletions, and finding data in constant time. Hashing is a key-to-address mapping process
- **Technical Term for Hashing**
- Data to be stored to an array → **Key**
- Array to store data → **hash table**
- The size of the array → **hash size**
- Formula to store data to the array → **hash function**
- The location of data in the array → **address**
- Using the hash function, multiple data are assigned to the same location in a hash table → **collision**

- **Hashing Functions**
  - **Division**
    H(key) = key % hashSize,
  - **Folding**
    H(key, C) =(floor(key/C) + key%C)%hashSize
    ** C is a constant
- **Collision Resolution**
  - **Separate Chaining:** use a list to keeps all elements that hash to the same value.
  - **Open Addressing**: Reallocate a new cell when there is a collision. Alternative cells are tried in succession until the empty cell is found
    $\boldsymbol{H(key, i) = \big(H(key) + f(i)\big)\%hashSize}$
    - Linear probing f(i)=i
    - Quadradic probling $f(i)=i^2$

The UML of a hashtable class used in this lab is as follows.

HashTable
H: int[] hsize: int hfunction: String openAddrType: String
HashTable(size: int, hfunction: String, openAddrType: String) fillHashTable(data: int): void linearProbe(data:int): void quadraticProbe(data:int): void search(data:int): int

Exercise 1 a. Complete the method SelectionSort. Print out the sequence when there is a change in the sequence. Test your method in the main method. Hint: use method `int findIndexSmallest (int [] A, int start, int end)` to find the index of the smallest at each round. Uncomment the codes in the main method for SelectionSort to check the answer.

Exercise 1 b. Complete the method InsertionSort. Print out the sequence when there is a change in the sequence. Test your method in the main method. Uncomment the codes in the main method for InsertionSort to check the answer.

Exercise 2. Hashing: Complete the method fillHashTable.

Exercise 3. Complete the method MergeSort. Print out the sequence when there is a change in the sequence. Test your method in the main method.

Exercise 4. Complete the method RecursiveQuickSort. Print out the sequence when there is a change in the sequence. Test your method in the main method.

Exercise 5 Hashing: Complete the method Search.