Tutorial 7: Sorting

CSC 212: Data Structures

King Saud University

Insertion sort

Insertion sort gradually builds the sorted array by putting each new key in its correct position.

```
public static void insertionSort(int[] A, int n) {
    for (int i = 1; i < n; i++) {
        int j = i;
        while (j > 0 && A[j - 1] > A[j]) {
            int tmp = A[j];
            A[j] = A[j - 1];
            A[j - 1] = tmp;
            j--;
        }
    }
}
```

Insertion sort

Example

$\Downarrow$ indicates i, $\uparrow$ indicates j.

$$\left(10, \underset{\uparrow}{\overset{\Downarrow}{3}}, 6, 14, 7\right)$$

Example

$\Downarrow$ indicates i, $\uparrow$ indicates j.

$$\left(10, \underset{\uparrow}{\overset{\Downarrow}{3}}, 6, 14, 7 \right)$$

$$\left(\underset{\uparrow}{3}, \overset{\Downarrow}{10}, 6, 14, 7 \right)$$

Insertion sort

Example

$\Downarrow$ indicates i, $\uparrow$ indicates j.

$$\left(10, \overset{\Downarrow}{\underset{\uparrow}{3}}, 6, 14, 7\right)$$

$$\left(\underset{\uparrow}{3}, \overset{\Downarrow}{10}, 6, 14, 7\right)$$

$$\left(3, 10, \overset{\Downarrow}{\underset{\uparrow}{6}}, 14, 7\right)$$

Example

$\Downarrow$ indicates i, $\uparrow$ indicates j.

$$\left(10, \overset{\Downarrow}{\underset{\uparrow}{3}}, 6, 14, 7\right)$$

$$\left(\underset{\uparrow}{3}, \overset{\Downarrow}{10}, 6, 14, 7\right)$$

$$\left(3, 10, \overset{\Downarrow}{\underset{\uparrow}{6}}, 14, 7\right)$$

$$\left(3, \underset{\uparrow}{6}, \overset{\Downarrow}{10}, 14, 7\right)$$

Insertion sort

Example

$\Downarrow$ indicates i, $\uparrow$ indicates j.

$$\left(10, \underset{\uparrow}{\overset{\Downarrow}{3}}, 6, 14, 7\right)$$

$$\left(\underset{\uparrow}{3}, \overset{\Downarrow}{10}, 6, 14, 7\right)$$

$$\left(3, 10, \underset{\uparrow}{\overset{\Downarrow}{6}}, 14, 7\right)$$

$$\left(3, \underset{\uparrow}{6}, \overset{\Downarrow}{10}, 14, 7\right)$$

$$\left(3, 6, 10, \underset{\uparrow}{\overset{\Downarrow}{14}}, 7\right)$$

Insertion sort

Example

$\Downarrow$ indicates i, $\uparrow$ indicates j.

$$\left(10, \overset{\Downarrow}{\underset{\uparrow}{3}}, 6, 14, 7\right)$$

$$\left(\underset{\uparrow}{3}, \overset{\Downarrow}{10}, 6, 14, 7\right)$$

$$\left(3, 10, \overset{\Downarrow}{\underset{\uparrow}{6}}, 14, 7\right)$$

$$\left(3, \underset{\uparrow}{6}, \overset{\Downarrow}{10}, 14, 7\right)$$

$$\left(3, 6, 10, \overset{\Downarrow}{\underset{\uparrow}{14}}, 7\right)$$

$$\left(3, 6, 10, 14, \overset{\Downarrow}{\underset{\uparrow}{7}}\right)$$

Insertion sort

Example

$\Downarrow$ indicates i, $\uparrow$ indicates j.

$$\left(10, \overset{\Downarrow}{\underset{\uparrow}{3}}, 6, 14, 7\right)$$

$$\left(\underset{\uparrow}{3}, \overset{\Downarrow}{10}, 6, 14, 7\right)$$

$$\left(3, 10, \overset{\Downarrow}{\underset{\uparrow}{6}}, 14, 7\right)$$

$$\left(3, \underset{\uparrow}{6}, \overset{\Downarrow}{10}, 14, 7\right)$$

$$\left(3, 6, 10, \overset{\Downarrow}{\underset{\uparrow}{14}}, 7\right)$$

$$\left(3, 6, 10, 14, \overset{\Downarrow}{\underset{\uparrow}{7}}\right)$$

$$\left(3, 6, 10, \underset{\uparrow}{7}, \overset{\Downarrow}{14}\right)$$

Insertion sort

Example

$\Downarrow$ indicates i, $\uparrow$ indicates j.

$$\left(10, \overset{\Downarrow}{\underset{\uparrow}{3}}, 6, 14, 7\right) \qquad \left(3, 6, 10, \overset{\Downarrow}{\underset{\uparrow}{14}}, 7\right)$$

$$\left(\underset{\uparrow}{3}, \overset{\Downarrow}{10}, 6, 14, 7\right) \qquad \left(3, 6, 10, 14, \overset{\Downarrow}{\underset{\uparrow}{7}}\right)$$

$$\left(3, 10, \overset{\Downarrow}{\underset{\uparrow}{6}}, 14, 7\right) \qquad \left(3, 6, 10, \underset{\uparrow}{7}, \overset{\Downarrow}{14}\right)$$

$$\left(3, \underset{\uparrow}{6}, \overset{\Downarrow}{10}, 14, 7\right) \qquad \left(3, 6, \underset{\uparrow}{7}, 10, \overset{\Downarrow}{14}\right)$$

Selection sort

Selection sort gradually builds the sorted array by finding the correct key for each new position.

```
public static void selectionSort(int[] A, int n) {
    for (int i = 0; i < n - 1; i++) {
        int min = i;
        for (int j = i + 1; j < n; j++) { // Search for the minimum
            if (A[j] < A[min])
                min = j;
        }
        // Swap A[i] with A [min]
        int tmp = A[i];
        A[i] = A[min];
        A[min] = tmp;
    }
}
```

Selection sort

Example

$\Uparrow$ indicates i, $\uparrow$ indicates min.

$$\left(\underset{\Uparrow}{10}, \underset{\uparrow}{3}, 6, 14, 7\right)$$

Selection sort

Example

$\Uparrow$ indicates i, $\uparrow$ indicates min.

$$\left(\underset{\Uparrow}{10}, \underset{\uparrow}{3}, 6, 14, 7\right)$$

$$\left(3, \underset{\Uparrow}{10}, \underset{\uparrow}{6}, 14, 7\right)$$

Selection sort

Example

$\Uparrow$ indicates i, $\uparrow$ indicates min.

$$\left(\underset{\Uparrow}{10}, \underset{\uparrow}{3}, 6, 14, 7\right)$$

$$\left(3, \underset{\Uparrow}{10}, \underset{\uparrow}{6}, 14, 7\right)$$

$$\left(3, 6, \underset{\Uparrow}{10}, 14, \underset{\uparrow}{7}\right)$$

Example

$\Uparrow$ indicates i, $\uparrow$ indicates min.

$$\left(\underset{\Uparrow}{10}, \underset{\uparrow}{3}, 6, 14, 7\right)$$

$$\left(3, \underset{\Uparrow}{10}, \underset{\uparrow}{6}, 14, 7\right)$$

$$\left(3, 6, \underset{\Uparrow}{10}, 14, \underset{\uparrow}{7}\right)$$

$$\left(3, 6, 7, \underset{\Uparrow}{14}, \underset{\uparrow}{10}\right)$$

Selection sort

Example

$\Uparrow$ indicates i, $\uparrow$ indicates min.

$$\left(\underset{\Uparrow}{10}, \underset{\uparrow}{3}, 6, 14, 7\right) \qquad \left(3, 6, 7, \underset{\Uparrow}{14}, \underset{\uparrow}{10}\right)$$

$$\left(3, \underset{\Uparrow}{10}, \underset{\uparrow}{6}, 14, 7\right) \qquad \left(3, 6, 7, 10, \underset{\Uparrow}{14}\right)$$

$$\left(3, 6, \underset{\Uparrow}{10}, 14, \underset{\uparrow}{7}\right)$$

Bubble sort

Bubble sort sorts the array by repeatedly swapping non-ordered adjacent keys. After each for loop iteration, the maximum is moved (or *bubbled*) towards the end.

```
public static void bubbleSort(int A[], int n) {
    for (int i = 0; i < n - 1; i++) {
        for (int j = 0; j < n - 1 - i; j++) {
            if (A[j] > A[j + 1]) {
                // Swap A[j] with A[j + 1]
                int tmp = A[j];
                A[j] = A[j + 1];
                A[j + 1] = tmp;
            }
        }
    }
}
```

Example

$$(10, 3, 6, 14, 9)$$

Bubble sort

Example

$$(10, 3, 6, 14, 9)$$
$$(3, 6, 10, 9, |14)$$

Bubble sort

Example

$$(10, 3, 6, 14, 9)$$
$$(3, 6, 10, 9, |14)$$
$$(3, 6, 9, |10, 14)$$

Bubble sort

Example

$$(10, 3, 6, 14, 9)$$
$$(3, 6, 10, 9, |14)$$
$$(3, 6, 9, |10, 14)$$

$$(3, 6, |9, 10, 14)$$

Bubble sort

Example

$$(10, 3, 6, 14, 9)$$
$$(3, 6, 10, 9, |14)$$
$$(3, 6, 9, |10, 14)$$
$$(3, 6, |9, 10, 14)$$
$$(3, |6, 9, 10, 14)$$

Merge sort

Merge sort is a divide-and-conquer algorithms to sort an array of n elements:

1. Divide the array into two equal parts.
2. Sort each part apart (recursively).
3. Merge the two sorted parts.

The key step in merge sort is merging two sorted arrays, which can be done in $O(n)$.

Example

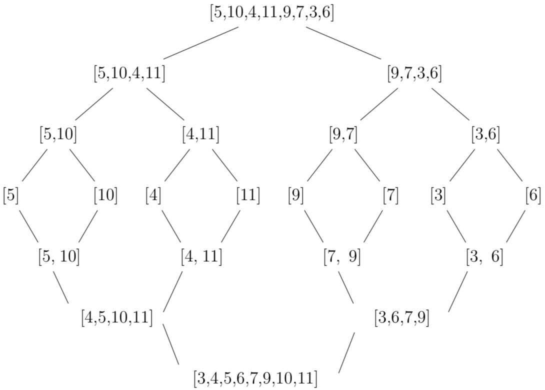
Given two arrays $B = \{1, 4, 6\}$ and $C = \{2, 3, 7, 8\}$, the result of merging B and C is $\{1, 2, 3, 4, 6, 7, 8\}$.

```
public static void mergeSort(int[] A, int l, int r) {
    if (l >= r)
        return;
    int m = (l + r) / 2;
    mergeSort(A, l, m); // Sort first half
    mergeSort(A, m + 1, r); // Sort second half
    merge(A, l, m, r); // Merge
}
```

```
private static void merge(int[] A, int l, int m, int r) {
    int[] B = new int[r - l + 1];
    int i = l, j = m + 1, k = 0;
    while (i <= m && j <= r)
        if (A[i] <= A[j])
            B[k++] = A[i++];
        else
            B[k++] = A[j++];
    if (i > m)
        while (j <= r)
            B[k++] = A[j++];
    else
        while (i <= m)
            B[k++] = A[i++];
    for (k = 0; k < B.length; k++)
        A[k + l] = B[k];
}
```

Example

Sort the array: $5, 10, 4, 11, 9, 7, 3, 6.$



Quick sort

Quick sort is another divide-and-conquer algorithms to sort an array of n elements:

1. Pick any element of the array and call it the pivot (the first element, or a randomly chosen element for example) .
2. Rearrange the array so that all elements before the pivot are less or equal the pivot, and all those after the pivot are greater or equal to the pivot (**partitioning**).
3. Recursively sort the part of the array before the pivot and the one after the pivot.

Quick sort

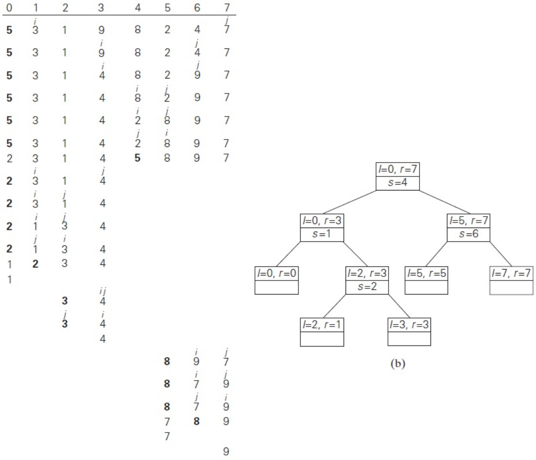
```
public static void quickSort(int[] A, int l, int r) {
    if (l < r) {
        int s = partition(A, l, r);
        quickSort(A, l, s - 1);
        quickSort(A, s + 1, r);
    }
}
```

Quick sort

```
private static int partition(int[] A, int l, int r) {
    int p = A[l], i = l + 1, j = r;
    while (i < j) {
        while (A[i] <= p && i < j)
            i++;
        while (A[j] > p && i < j)
            j--;
        int tmp = A[i];
        A[i] = A[j];
        A[j] = tmp;
    }
    int s;
    if (A[i] <= p) s = i; else s = i - 1;
    int tmp = A[l];
    A[l] = A[s];
    A[s] = tmp;
    return s;
}
```

Example

Sort the array:
$5, 3, 1, 9, 8, 2, 4, 7.$



(a) (b)

Bucket sort

Bucket sort is a non-comparison-based sorting algorithm that can be used to sort positive integer keys as follows:

1. Create an array of k "buckets", initially all empty.
2. Put each element of A in its bucket.
3. Sort each non-empty bucket (using insertion sort for example).
4. Concatenate all sorted buckets.

Bucket sort

```
public static void bucketSort(int[] A, int n, int k) {
    // Create empty buckets
    List<Integer>[] buckets = new List[k];
    for (int b = 0; b < k; b++)
        buckets[b] = new LinkedList<Integer>();
    // Put each element in its bucket
    int max = max(A, n);
    max++;
    for (int i = 0; i < n; i++)
        buckets[k * A[i] / max].insert(A[i]);
    // Sort and concatenate buckets
    int i = 0;
    for (int b = 0; b < k; b++) {
        int[] B = sort(buckets[b]);
        for (int j = 0; j < B.length; j++)
            A[i++] = B[j];
    }
}
```

King Saud University

Example

Let $A = \{6, 22, 3, 15, 12, 25, 9\}$, and $k = 3$

- Create an array of 3 empty buckets: Bucket 0 (for $0 \leq A[i] < 10$), Bucket 1 (for $10 \leq A[i] < 20$) and Bucket 2 (for $20 \leq A[i] < 30$).

Bucket 0	Bucket 1	Bucket 2

- Assign elements to buckets (we use a linked list to implement buckets):

Bucket 0	Bucket 1	Bucket 2
$6 \rightarrow 3 \rightarrow 9$	$15 \rightarrow 12$	$22 \rightarrow 25$

King Saud University

Example

- Sort buckets:

Bucket 0	Bucket 1	Bucket 2
$3 \to 6 \to 9$	$12 \to 15$	$22 \to 25$

- Concatenate all sorted buckets back in A:

$$A : \{3, 6, 9, 12, 15, 22, 25\}.$$

Counting sort

Counting sort is a simple efficient sorting algorithm that does not use comparison. It is used when the keys are small positive integers.

Counting sort

```
public static void countingSort(int[] A, int n, int m) {
    int[] counts = new int[m];
    for (int j = 0; j < m; j++)
        counts[j] = 0;
    for (int i = 0; i < n; i++) // Count frequency
        counts[A[i]]++;
    for (int j = 1; j < m; j++) // Compute prefix sum
        counts[j] += counts[j - 1];
    int[] B = new int[n];
    for (int i = n - 1; i >= 0; i--) { // Put elements in correct order in B
        B[counts[A[i]] - 1] = A[i];
        counts[A[i]]--;
    }
    for (int i = 0; i < n; i++) // Copy back B to A
        A[i] = B[i];
}
```

Counting sort

Example

Consider the array $A = \{3, 9, 3, 4, 1, 5\}$. Since all values are less than 10, we create an array of size 10 that counts the frequency of each key in A (any size $> max(A)$ will work):

1. Create an array called $counts$ and initialize it to 0: $counts = \{0, 0, 0, 0, 0, 0, 0, 0, 0, 0\}$
2. Go through A and increment $counts[A[i]]$: $counts = \{0, 1, 0, 2, 1, 1, 0, 0, 0, 1\}$
3. Compute the prefix sum of $counts$: $counts = \{0, 1, 1, 3, 4, 5, 5, 5, 5, 6\}$
4. Notice that $counts[A[i]] - 1$ contains the correct index of the last occurrence of $A[i]$. For example, $counts[A[2]] - 1 = 2$, which means the second 3 in A should be in position 2. To get the position of the previous 3, we just decrement $counts[A[i]]$.
5. Create an array B of the same size as A. Move through A **backwards** and assign each key in A to its correct position in B: $B[counts[A[i]] - 1] = A[i]$, then decrement $counts[A[i]] - -$: $B = \{1, 3, 3, 4, 5, 9\}$.
6. Finally, copy B to A.

Radix sort

- Radix sort is a non comparison-based sorting algorithm that can be used to sort positive integer keys.
- The algorithm considers the digits of the keys one by one and sorts the keys each time according to the selected digit.
- At each step the keys are sorted using simple sorting algorithm such as counting sort.
- Radix sort can also be used to sort strings.

Radix sort

```
public static void radixSort(int[] A, int n, int b) {
    int[] B = new int[n];
    int dv = 1;
    while (true) {
        boolean done = true;
        for (int i = 0; i < n; i++) {
            B[i] = (A[i] / dv) % b; // Extract digit
            if (B[i] != 0) done = false;
        }
        if (done) break;
        int[] index = countingSortIndex(B, n, b);
        for (int i = 0; i < n; i++)
            B[index[i]] = A[i];
        for (int i = 0; i < n; i++)
            A[i] = B[i];
        dv *= b;
    }
}
```

Radix sort

CountingSortIndex is the same as CountingSort, except that it returns the index instead of sorting A.

```
private static int[] countingSortIndex(int[] A, int n, int m) {
    int[] counts = new int[m];
    for (int j = 0; j < m; j++)
        counts[j] = 0;
    for (int i = 0; i < n; i++)
        counts[A[i]]++;
    for (int j = 1; j < m; j++)
        counts[j] += counts[j - 1];
    int[] index = new int[n];
    for (int i = n - 1; i >= 0; i--) {
        index[i] = counts[A[i]] - 1;
        counts[A[i]]--;
    }
    return index;
}
```

Radix sort

King Saud University

Example

We want to sort the array $A = \{27, 325, 72, 6, 150, 72\}$. We are going to apply radix search by considering decimal digits at each iteration. Hence, our base $b = 10$. We can choose other bases: $b =$ 2, 4 or 256 for example.

1. Sort keys according to first digit:
   $\{7, 5, 2, 6, 0, 2\} \rightarrow \{150, 72, 72, 325, 6, 27\}$.
2. Sort keys according to second digit:
   $\{5, 7, 7, 2, 0, 2\} \rightarrow \{6, 325, 27, 150, 72, 72\}$.
3. Sort keys according to third digit:
   $\{0, 3, 0, 1, 0, 0\} \rightarrow \{6, 27, 72, 72, 150, 325\}$.

Remark

How to extract digits?: 1st: $A[i]\%10$, 2nd: $(A[i]/10)\%10$, 3rd: $(A[i]/100)\%10$, etc.