Notes for ECE2810J

Tianzong Cheng

Contents

1. Comparison Sort

1.1. Bubble Sort

```
// Function to perform Bubble Sort
void bubbleSort(int arr[], int n) {
  for (int i = 0; i < n - 1; i++) {
    // Flag to optimize the algorithm
    bool swapped = false;

    // Last i elements are already in place, so we don't need to check them
    for (int j = 0; j < n - i - 1; j++) {
      if (arr[j] > arr[j + 1]) {
        // Swap arr[j] and arr[j+1]
        int temp = arr[j];
        arr[j] = arr[j + 1];
        arr[j + 1] = temp;
        swapped = true;
      }
    }

    // If no two elements were swapped in inner loop, the array is already
    // sorted
    if (!swapped) {
      break;
    }
  }
}
```

Note that the last `i` elements don’t need to be checked: `for (int j = 0; j < n - i - 1; j++)` .

1.2. Selection Sort

```
// 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;
  }
}
```

1.3. Insertion Sort

Review this before exam!

```
// Function to perform Insertion Sort
void insertionSort(int arr[], int n) {
  for (int i = 1; i < n; i++) {
    int key = arr[i];
    int j = i - 1;

    // Move elements of arr[0..i-1], that are greater than key,
    // to one position ahead of their current position
    while (j >= 0 && arr[j] > key) {
      arr[j + 1] = arr[j];
      j--;
    }
    arr[j + 1] = key;
  }
}
```

1.4. Merge Sort

```
// 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);

  // Copy data to temporary arrays L[] and R[]
  for (int i = 0; i < n1; i++) {
    L[i] = arr[l + i];
  }
  for (int i = 0; i < n2; i++) {
    R[i] = arr[m + 1 + i];
  }

  // Merge the temporary arrays back into arr[l..r]
  int i = 0; // Initial index of first subarray
  int j = 0; // Initial index of second subarray
  int k = l; // Initial index of merged subarray

  while (i < n1 && j < n2) {
    if (L[i] <= R[j]) {
      arr[k] = L[i];
      i++;
    } else {
      arr[k] = R[j];
      j++;
    }
    k++;
  }

  // Copy the remaining elements of L[], if there are any
  while (i < n1) {
    arr[k] = L[i];
    i++;
    k++;
  }

  // Copy the remaining elements of R[], if there are any
  while (j < n2) {
    arr[k] = R[j];
    j++;
    k++;
  }
}
```

```
// 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
    mergeSort(arr, l, m);
    mergeSort(arr, m + 1, r);

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

1.5. Quick Sort

```
// 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

  for (int j = low; j <= high - 1; j++) {
    // If the current element is smaller than or equal to the pivot
    if (arr[j] <= pivot) {
      i++;
      // Swap arr[i] and arr[j]
      std::swap(arr[i], arr[j]);
    }
  }

  // Swap arr[i+1] and arr[high] (or the pivot)
  std::swap(arr[i + 1], arr[high]);
  return (i + 1);
}

// 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
    quickSort(arr, low, pi - 1);
    quickSort(arr, pi + 1, high);
  }
}
```