Algorithms and Data Structures (ADS2)

Laboratory Sheet 3

This lab sheet contains material based on Lectures 6-7. This exercise is not assessed but should be completed to gain sufficient experience in the implementation of several variants of QUICKSORT.

Exercise

This exercise has two parts. You are asked to implement four variants of the quicksort algorithm in Java in the first part, and a creative problem involving the median-of-three partition scheme in the second part.

Part 1

Implement the following algorithms in Java:

a) The QUICKSORT algorithm introduced in Lecture 7 (slide 14), including procedure PARTITION implementing right-most element pivot selection (slide 13).

```
private static int partition(int a[], int p, int r){
  int x = a[r];
  int i = p - 1;
  for (j = p; j < r; j++){
    if (a[j] <= x){
      i++;
      swap(a, i, j);
    }
  }
  swap(a, i+1, r);
  return i + 1;
}

public static void sort(int a[], int p, int r){
  if (r <= p) return;
  int q = partition(a, p, r);
  sort(a, p, q-1);
  sort(a, q+1, r);
}
```

b) A variant of QUICKSORT in which the pivot is selected randomly in PARTITION. Compare the performance of this implementation against standard QUICKSORT on `bad.txt`.

```
public static void rSort(int a[], int p, int r){
  if (r <= p) return;
  Random random = new Random();
  swap(a, random.nextInt(r - p) + p, r);
  int q = partition(a, p, r);
  rSort(a, p, q-1);
  rSsort(a, q+1, r);
}
```

c) A variant of QUICKSORT which returns without sorting subarrays with fewer than k elements and then uses INSERTION-SORT to sort the entire nearly-sorted array (Lecture 7 slide 26).

```
private static void kSort(int a[], int p, int r, int k){
  if (r - p <= k) return;
  int q = partition(a, p, r);
  kSort(a, p, q-1);
  kSort(a, q+1, r);
}

public static void sortCutOff(int a[], int p, int r, int k){
  ksort(a, p, r, k);
  InsertionSort.sort(a, p, r);
}
```

d) 3-WAY-QUICKSORT (Lecture 7, slide 27). Compare the performance of this implementation against the previous variants of QUICKSORT on `dutch.txt`.

```
public static void sort3Way (int a[], int l, int r){
  if (r <= l) return;
  int v = a[r];
  int i = l-1, j = r, p = l-1, q = r, k;
  for (;;){
    while (a[++i]<v);
    while (v < a[--j]) if (j == l) break;
    if (i >= j) break;
    swap(a, i, j);
    if (a[i] == v) {p++; swap(a, p, i);}
    if (v == a[j]) {q--; swap(a,q,j);}
  }
  swap(a, i, r);
  j = i - 1;
  i = i + 1;
  for (k = l; k <= p; k++,j--) swap(a, k, j);
  for (k = r-1; k >=q; k--,i++) swap(a, k, i)
  sort3Way(a, l, j);
  sort3Way(a, i, r);
}
```

Part 2

a) Implement a variant of QUICKSORT adopting the median-of-3 partitioning scheme.

```
public static void sortMedian3(int a[], int p, int r){
  if (r <= p) return;
  swap(a, (p+r)/2, r-1);
  if (a[r-1] < a[p]) swap (a, p, r-1);
  if (a[r] < a[p]) swap (a, p, r);
  if (a[r] < a[r-1]) swap (a, r-1, r);
  int q = partition(a, p+1, r-1);
  sortMedian3(a, p, q-1);
  sortMedian3(a, q+1, r);
}
```

b) Implement in Java an algorithm to generate pathological input arrays of length n for the median-of-3 QUICKSORT algorithm, that is instances that will require quadratic time to be sorted. Assume no duplicates in the input array.

There are several possible ways to implement this. For example, the median-of-3 scheme can easily be beaten if **at every recursive call** the median of left, middle and right is the second highest value.