Birla Institute of Technology & Science, Pilani
2nd Semester 2016-17 - CS F211 – Data Structures and Algorithms

Lab 5 – 16th Feb 2017
Topics – Sorting and Practical Performance of Sorting Algorithms

1. Implement a QuickSort test-bed with multiple partitioning and pivot selection algorithms.

Implement (recursive version of) QuickSort in file **qs.c** which includes header files **part.h** and **pivot.h** available in the local directory:

/*part.h*/

/* Implements partitioning algorithm */
/* Precondition: **Ls** is an array indexed from **lo** to **hi** ; **pInd** is the index, in **Ls**, of the pivot */
/* returns **pInd** , the rank of the pivot */
/* Postcondiiton:
(forall j: lo<=j<pInd --> Ls[j]<=Ls[pInd]) AND (forall j: pInd<j<=hi --> Ls[j]>Ls[pInd]) */
int part(int Ls[], int lo, int hi, int pInd);

/*pivot.h*/

/* Implements pivot selection algorithm */
/* Precondition: Ls is an array inde xed from **lo** to **hi** */
/* returns **pInd,** the index, in **Ls**, of the chosen pivot */
int pivot(int Ls[], int lo, int hi);

Instrument QuickSort such that number of recursive calls and recursion depth are counted (separately, using global variables) and printed.

2. Implement two different versions of partitioning:

a) Implement procedure **part** in file **part1.c** such that the partitioning is done from both ends. (*see Figure 1. below*: Hoare's partitioning). Compile **part1.c** using the ***gcc –c*** command option to generate the **part1.o** object file.

b) Implement procedure **part** in file **part2.c** such that the partitioning is done from one end. (*see Figure 2. below*: Locality-aware partitioning). Compile **part2.c** using the ***gcc –c*** command option to generate the **part2.o** object file.

3. Implement two different versions of pivot selection:

a) Implement procedure **pivot** in file **pivot1.c** such that the pivot value is the median of the first, the last, and the middle indexed values i.e. median of **Ls[lo]**, **Ls[hi]**, and **Ls[mid]** where **mid=(lo+hi)/2**. Compile **pivot1.c** using the ***gcc –c*** command option to generate the **pivot1.o** object file.

b) Implement procedure **pivot** in file **pivot2.c** such that the pivot value is located at an index chosen *uniformly randomly* in the range **lo..hi**. Compile **pivot2.c** using the ***gcc –c*** command option to generate the **pivot2.o** object file.

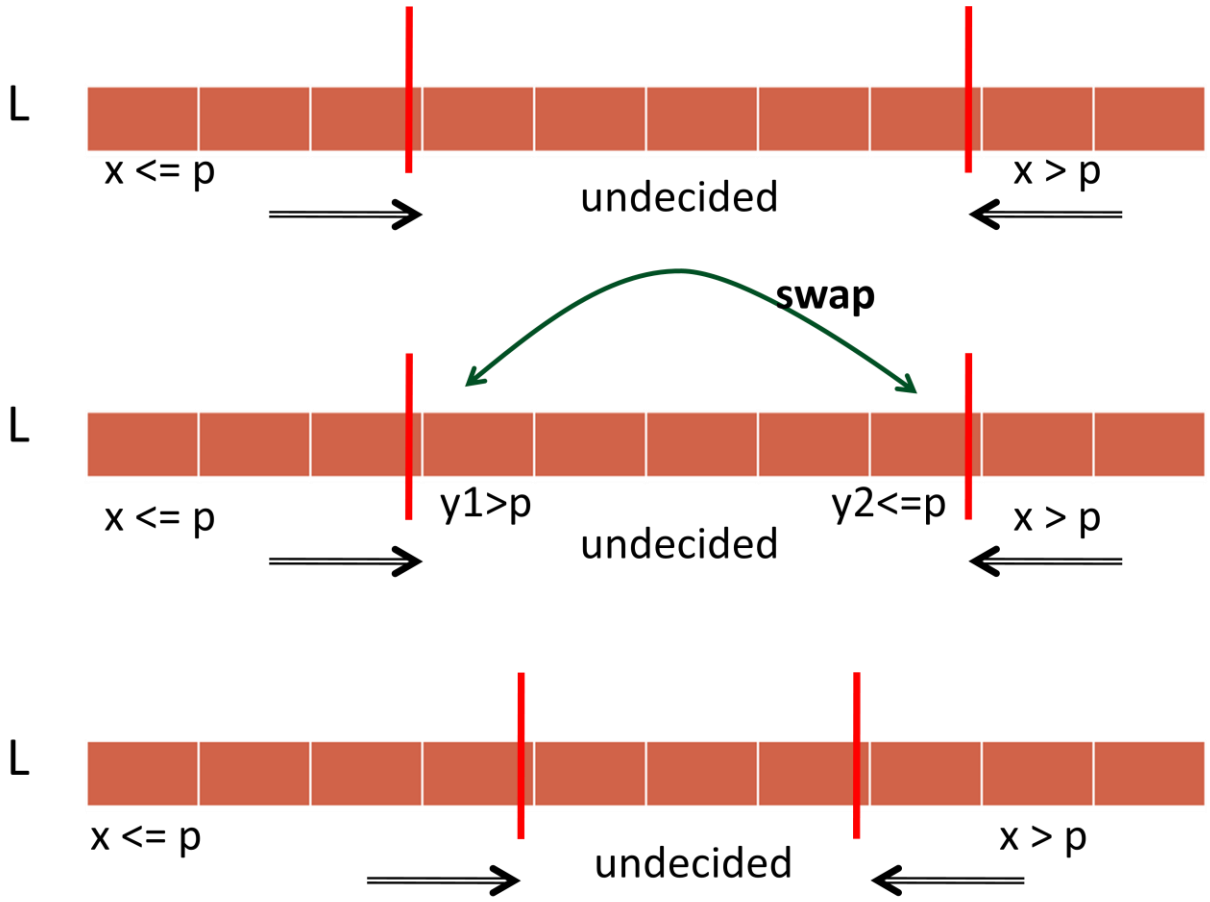

Figure 1: Hoare's partitioning – Single Iteration: swapping maintains invariant

==========

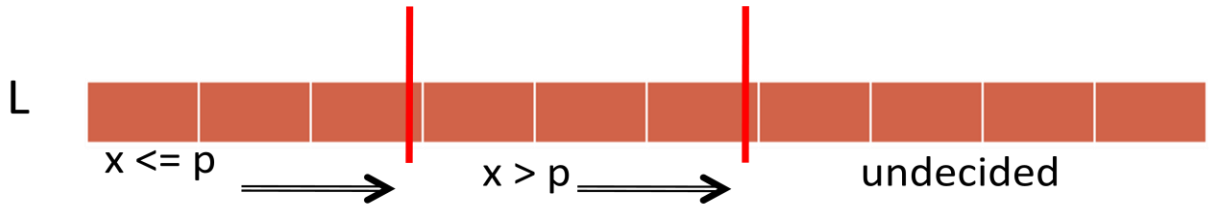

Figure 2: Locality-aware partitioning: invariant

4. Compile **qs.c** with different combinations of partition procedure and pivot selection procedure to generate different executables, say, **qsHoareM3**, **qsLocalM3**, **qsHoareRand**, and **qsLocalRand**. Executable files may be named by using the ***–o*** option in ***gcc***.

Use following structure for implementation (sort on maximum of two marks for each student):

Student:

Name	: single word (at most 20 characters)
Marks1	: double
Marks2	: double

Input Format:
0 followed by an integer N equal to number of student records.
N lines follow, each line contains one student record (Student_name Marks1 Marks2)

Output Format: N student records sorted according to the max. of the two marks.

Data Generator:
You may use following code to generate input data records:

```
/* random_generator.c */
#include <stdio.h>
#include <assert.h>
#include <stdlib.h>
void rand_str(char *dest, size_t length) {
      char charset[] = "abcdefghijklmnopqrstuvwxyz"
                       "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
      while (length-- > 0) {
      size_t index = (double) rand() / RAND_MAX * (sizeof charset - 1);
      *dest++ = charset[index];
      }
      *dest = '\0';
}
double rand_double(double min, double max)
{
      return (double)rand()/(double)RAND_MAX * (max - min) + min;
}
int main(int argc, char *argv[])
{
      int N = 10;
      assert(argc == 2);
      int i = atoi(argv[1]);
      char *str = (char*) malloc (sizeof(char) * N);
      assert(str != NULL);
      //rand_str(str, N);
      while(i--)
      {
            rand_str(str, N);
            printf("%s\t%lf\t%lf\n",       str,       rand_double(0,100),
            rand_double(0,100));
      }
      return 0;
}
/* End of random_generator.c */
```