12-13. Sorting and Searching

msdb@korea.ac.kr

Agenda

- Instruction
- Merge sort
- Hash searching

Instruction

- Make a zip file named “studentid” that includes one folder and source codes.
- Make sure your codes can be properly compiled.
- Do not submit whole solution file.

Merge Sort

- Implement a function which sorts integer values in an array using merge sort algorithm
  - Algorithms: Use merge sort algorithm to sort values.

  - `void merge(int arr[], int low, int mid, int high)`
  - `void mergeSort(int arr[], int low, int high)`

MAIN

```c
#include <stdio.h>
#include <stdlib.h>
#define N 10

void printArr(int arr[], const size_t size)
{
	for (int i = 0; i < size; i++)
	{
		printf("%d ", arr[i]);
	}
	printf("\n");
}

void merge(int arr[], int low, int mid, int high)
{
	// merge body
}

void mergeSort(int arr[], int low, int high)
{
	// merge sort body
}
```

```c
int main()
{
	int values[N] = { 1,0,6,7,3, 9,6,6,2,8 };

	printArr(values, N);
	mergeSort(values, 0, N - 1);
	printArr(values, N);

	return 0;
}
```

Hash Search

- Implement a function which search an index of an integer value in an array using hash search algorithm
  - Algorithms: Use any hash search algorithm to search value.

MAIN

```c
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#ifdef _MSC_VER
// Windows
#include <Windows.h>
#else
// Linux
#include <time.h>
#endif

#define N 1000000
#define MAX_VALUE 10000

void runSearch(void* pList, const size_t size, const int target, int* pLoc, bool (*searchFunc)(void*, const size_t, const int, int*))
{
	LARGE_INTEGER freq;
	LARGE_INTEGER beginTime;
	LARGE_INTEGER endTime;

	QueryPerformanceFrequency(&freq);
	QueryPerformanceCounter(&beginTime);
	bool result = searchFunc(pList, size, target, pLoc);
	QueryPerformanceCounter(&endTime);

	double duringTime = (double)(endTime.QuadPart -
		beginTime.QuadPart) / (double)freq.QuadPart;

	printf("Execution time: %.10lf ms\n", duringTime * 10e3);
	if (result)
	printf("Found %d at %d\n", target, *pLoc);
	else
	printf("Cannot found %d\n", target);
}

bool seqSearch(void* pList, const size_t size, const int target, int* pLoc)
{
	unsigned int i = 0;
	int* arr = (int*)pList;
	while (i < size && arr[i] != target)
	{
		i++;
	}
	*pLoc = i;

	return (arr[i] == target);
}

int getRand()
{
	return rand() % MAX_VALUE;
}
```

MAIN

```
void buildHash(void* pList, size_t listSize, void* pHash,
size_t* hashSize)
{
      // buildHash body
}

bool hashSearch(void* pHash, const size_t size, const int
target, int* pLoc)
{
      // hashSearch body
}

void deleteHash(void* pHash, const size_t size)
{
      // deleteHash body
}
```

```
int main()
{
      static int arr[N] = { 0 };

      srand(2019);
      for (int i = 0; i < N; i++)
      {
           arr[i] = getRand();
      }

      int target = getRand();
      int loc = -1;

      size_t hashSize = 0;
      // Make pHash
      buildHash(arr, N, (void*)pHash, &hashSize);

      runSearch(arr, N, target, &loc, seqSearch);

      runSearch(pHash, hashSize, target, &loc, hashSearch);

      deleteHash(pHash, hashSize);

      return 0;
}
```