CSE3081 (2반): 알고리즘 설계와 분석
<프로그래밍 숙제 3>

담당 교수: 임 인 성

2017년 11월 3일

마감: 11월 26일 일요일 오후 11시 정각
제출물, 제출 방법, LATE 처리 방법 등: 조교가 과목 게시판에 공고할 예정임.

목표: 이번 숙제에서는 수업 시간에 배운 선택 (selection)과 정렬 (sorting) 방법을 구현하여 이론적으로 배운 내용과 실제로 구현한 결과를 비교하여 본다. 특히 교과서적인 quick sort 방법의 속도 향상을 위하여 속도 최적화 기법을 적용한 후 분석하여봄으로써 고급 소프트웨어 개발능력을 배양토록 한다.

1. 본 숙제에서 다룰 배열 데이터의 각 원소는 다음과 같이 32바이트의 레코드 구조로 구성되어 있으며, 선택 또는 정렬을 해야할 기준이 되는 key 값은 아래와 같이 `ELEMENT` 타입의 배열 원소의 주소 `x`에 대한 매크로 함수를 사용하여 접근할 수 있다. 해당 내용은 본 숙제에서 제공하는 `my_types.h` 파일에 정의되어 있다.

```
typedef struct {
    unsigned int key; float score; double other; char name[16];
} ELEMENT;

#define ELEMENT_KEY(x) ((x)->key)
```

   선택 또는 정렬 방법 구현 시 원소 값을 이동시켜야할 경우 아래의 함수를 사용하라.

```
#include <string.h>
void *memcpy(void *dest, const void *src, size_t n);
```

2. **(구현 요구 사항)** 본 숙제에서 여러분이 구현하는 각 함수들은 각각 동일한 이름을 가지는 파일 `selection_rec.cpp`, `qsort.cpp`, `insertion_sort.cpp`, `heap_sort.cpp`, `quick_sort.cpp`, 그리고 `quick_sort_opt.cpp`에 저장하라 (조교는 숙제 채점 시 자신의 Visual Studio 프로젝트에 위 다섯 개의 파일을 복사한 후 자신의 주 함수에서 각 함수를 차례대로 호출하며 구현 내용과 수행 결과를 확인할 예정임. ← 본 숙제에서 제공하는 예제 프로그램 참고). 특히 각 함수들은 정상적으로 원하는 계산이 완료되었으면 `0` 값을 아니면 그 외의 값을 리턴해주어야 한다 (비정상적으로 종료할 경우 에러 형태에 따른 번호를 리턴해주면 되나, 본 숙제에서는 적절한 error code의 사용은 채점의 대상이 아님).

   (a) 먼저 강의자료 103쪽의 재귀적인 선택 알고리즘을 다음과 같은 프로토타입의 재귀 함수로 구현하라. 이 함수는 `data[left]`부터 `data[right]`까지 `n` (`= right-left+1`)개의 원소들 중 key 값의 크기가 `k`번째인 원소를 찾아 그 원소의 인덱스 (원래의 입력 배열에서의 인덱스) `*index`를 리턴해주어야 함.

```
int SELECTION_REC(ELEMENT data[], int left, int right, int k,
                  int *index);
```

(b) 다음과 같은 프로토타입을 가지는 정렬 방법을 구현하라. 이 함수들은 `data[left]`부터 `data[right]`까지 `n` (= `right-left+1`)개의 원소들을 `key` 값에 대하여 non-decreasing order로 정렬해주어야 함.

i. **Insertion sort:** 전형적인 insertion sort 방법을 구현함.

```
int INSERTION_SORT(ELEMENT data[], int left, int right);
```

ii. **Heap sort:** 전형적인 heap sort 방법을 구현함.

```
int HEAP_SORT(ELEMENT data[], int left, int right);
```

iii. **qsort:** 이 함수는 직접 입력 데이터를 정렬하지 않고 Standard C library가 제공하는 `qsort()` 함수를 호출하여 원하는 정렬 계산을 수행함.

```
int QSORT(ELEMENT data[], int left, int right);
```

iv. **Quick sort:** 자신만의 효과적인 pivot strategy를 사용하여 pivot 원소를 선택하는 재귀적인 방식의 전형적인 quick sort 방법을 구현함. Pivot 원소 선택은 가장 보편적인 방식대로 배열 데이터 제일 앞 (인덱스 `left`), 가운데 (인덱스 `(left+right)/2`), 제일 뒤 (인덱스 `right`)에 있는 세 원소들 중앙값을 가지는 원소를 pivot 원소로 사용하여도 좋고, 더 낳은 방법이 있으면 자신이 원하는 방식을 사용하여도 됨.

```
int QUICK_SORT(ELEMENT data[], unsigned int left,
                    int right);
```

v. **Optimized quick sort:** 바로 위에서 구현한 방법을 확장하여 최적화된 quick sort 방법을 구현하라. 최적화 기법으로는 기본적으로 강의자료 70쪽에 기술한 insertion sort와 결합하는 방법과 80쪽의 partition 후 길이가 짧은 쪽에 대해서만 recursion을 적용하는 방법 등을 고려할 수 있으며, 그 외에의 기법을 적용하여도 무방함. 여기서 최적화란 속도 향상을 의미함.

```
int QUICKSORT_OPT(ELEMENT data[], unsigned int left,
                       int right);
```

3. **(실험)** 각 구현 방법의 성능을 평가하기 위하여 $n = 1024$, 4096, 16384, 65536, 262144, 1048576, $\cdots$ (또는 메모리가 허용하는 범위 내에서 최대 크기까지) 각 구현 방법의 시간측정에 적합한 적절한 크기들을 선택하여 아래에 기술하는 세 가지 특성을 가지는 입력 데이터들을 생성한 후 실험하라. 본 숙제에서는 입력 데이터는 binary format으로 파일에 저장하야 하며, 그 형식은 먼저 첫 네 바이트에는 배열 데이터의 원소의 개수 `n`이 `int` 형식으로 저장하고, 이후 `sizeof(ELEMENT)` 바이트 씩 `n`개의 원소의 내용이 순서대로 저장되도록 하라. (← 본 숙제에서 제공하는 샘플 데이터 참고)

(a) **Entirely random:** 배열 원소의 key 값으로 적절한 범위의 random number를 n개 생성하여 만든 데이터. (← 본 숙제에서 제공하는 데이터 생성 프로그램 참조)

(b) **Descending:** 배열 원소의 key 값이 $n-1, n-2, n-3, \cdots, 1, 0$와 같이 정반대로 정렬되어 있는 배열 데이터.

(c) **Few swaps:** 배열 원소의 key 값이 $0, 1, 2, \cdots, n-1$과 같이 정렬된 상태에서 int($\sqrt{n}$)개의 random number pair (i, j)를 생성하여 i번째 원소와 j번째 원소를 서로 교환함 (이때 $|i-j| < \sqrt{n}$의 조건을 만족하도록 할 것).

4. **(보고서 기술)** 다음의 내용을 포함하여 본인의 구현 내용 특이 사항, 실험 방법, 성능 측정 결과, 그리고 분석 내용 등을 보고서에 명료하게 기술하라.

- 다음의 예처럼 실험에 사용한 CPU의 속도 및 메인 메모리의 용량 등의 실험 환경을 기술하라 (프로그램 수행에 충분한 크기의 메모리를 장착한 컴퓨터를 사용할 것).

```
OS: Window 10 Education
CPU: Intel(R) Core(TM)2 Duo CPU E6850 @ 3.00GHz 2.99Ghz
RAM: 4.00GB
Compiler: Visual Studio 2017 Release Mode
```

- 자신이 선정한 실험 데이터의 각각에 대하여 각 구현 함수의 수행 시간을 측정한 후 이를 적절한 형태의 테이블에 그 결과를 요약하라. 시간 측정은 본 수업에서 제시한 방법을 사용하며, 실험은 각 상황에 대하여 최소 다섯 번 이상 실험을 한 후 적절히 평균 시간을 취할 것.

- 과연 수행 시간과 이 방법들의 이론적인 시간 복잡도간에 깊은 연관성을 발견할 수 있는가?
- 이때 서로 다른 종류의 데이터에 대하여 수행 시간에 차이가 있는가? 있다면 그 이유는 무엇으로 추정되는가?
- 과연 quick sort 방법은 그 이름이 의미하는 바와 같이 정말 빠른가?
- 과연 어떤 부류의 데이터에 대해 insertion sort 방법의 수행 시간이 heap sort 또는 quick sort 방법에 비해 빠르거나 그리 느리지 않은가? 과연 insertion sort 방법이 거의 선형적인 시간에 작동하는 경우를 발견하였는가?
- n 값이 작을 경우 굳이 quick sort 방법을 사용할 필요없이 insertion sort 방법을 사용해도 크게 문제가 되지 않는다. 과연 이는 어느 정도 범위의 n 값까지일지 **Entirely random** 데이터에 대한 실험을 통하여 자신의 값을 제시하라.
- 입력 데이터의 크기 n에 quick sort 방법을 수행할 때 재귀적으로 생성되는 트리의 깊이가 어느 정도 되는지 각 리프 노드의 레벨에 대한 최소/최대/평균을 적절히 크기가 큰 **Entirely random** 데이터에 대한 실험을 통하여 자신의 값을 제시하라. 평균적으로 최악의 경우에 해당하는 $O(n)$에 가까운지 아니면 수업 시간에 이론적으로 증명한 바와 같이 $O(n \log n)$에 가까운지 실험적으로 분석할것.
- 각 방법을 최대한 효율적으로 구현하기 위하여 어떤 노력을 기울였는가?
- 기타 이번 숙제를 하면서 적용한 창의적인 방법이나 경험적으로 알게된 중요 사항을 기술하라.

5. **(속도 최적화 컨테스트)** 조교는 여러분이 구현한 함수 `QUICKSORT_OPT()` 함수에 대하여 적절히 크기가 큰 **Entirely random** 데이터를 사용하여 수행 시간을 측정한 후 속도 순으로 상위 10% – 20% 정도 (정확한 숫자는 상대적인 상황에 따라 추후 결정)의 학생에게 100점 만점 대비 최대 20점에 해당하는 추가 점수를 부여할 수 있다.

   - 이를 위하여 조교는 시간측정에 사용할 PC의 사양과 실험 데이터의 크기를 공지할 예정임.
   - 추가적인 점수를 받는 학생의 코드에 대해서는 특별히 자세한 검토가 있을 예정임.
   - 이 함수 구현 시, 2학년 2학기 컴퓨터공학 전공자가 생각할 수 있는 최적화 기법의 적용을 권장한다 (인터넷에서 "C/C++ Code Optimization" 주제로 검색을 해볼 것). 이러한 노력을 인정 받기 위해서는 보고서에 자신이 적용한 최적화 노력에 대하여 명확히 기술할 것.

- **[주의]**

   1. 자신의 작성한 함수의 수행은 본 숙제에서 제공하는 예제 프로그램을 참조하라.
   2. 본인의 결과를 `HW1_S2016****.{hwp,doc,txt}`와 같은 이름의 보고서에 기술하여 프로그램 및 데이터와 함께 제출하라 (파일명에 학번을 기술 또는 조교가 공지하는 방식으로 제출).
   3. 조교는 채점 시 여러분의 원시 코드를 확인한 후 자신의 명령어 파일과 입력 데이터를 사용하여 여러분의 프로그램이 정확한 값을 계산하는지 기계적으로 확인할 예정임.
   4. 숙제 제출 기간 동안 조교가 본 숙제와 관련하여 중요한 공지 사항을 게시판에 올릴 수 있으니 항상 수업 게시판을 확인하기 바람.
   5. 제출 화일에서 바이러스 발견 시 **본인 점수 X (-1)**이고, 다른 사람의 숙제를 복사할 경우 **관련된 사람 모두에 대하여 만점 X (-10)**임.