본 PSet 은 저의 강의 경험과 학생들의 의견 및 Stanford CS106 과 Harvard CS50 같은 강의에서 수집된 자료를 토대로 작성되었습니다.
본 PSet 에 문제점이나 질문 혹은 의견이 있다면, 저의 이메일(idebtor@gmail.com)로 알려 주시면 강의 개선에 많은 도움이 되겠습니다.

PSet: Profiling

목차

과제 수행 목적

이 프로젝트의 목적은 특정 조건에서 각 알고리즘의 성능을 측정하여 분석의 정확도를 경험적으로 검증하는 것입니다. 성능 측정 또는 프로그램 프로파일링은 다양한 수준의 세분화(granularity) 및 측정을 통해 알고리즘 성능에 대한 자세한 경험적 데이터를 제공합니다.
"프로파일링"은 프로그램의 공간(메모리) 복잡도 혹은 시간 복잡도, 특정 명령의 사용량 또는 함수 호출의 빈도 및 지속 시간 등을 측정합니다. 특수 프로파일링 도구만큼 정확하지는 않더라도 프로그램 실행에 따라 출력된 경과 시간을 사용합니다. 입력 데이터의 크기가 작으면 클럭 간격이 너무 커서 실행 시간을 측정할 수 없으므로 모든 시간이 0.0000 으로 출력될 수 있습니다. 이런 경우에는 다양한 데이터 또는 추가적인 코드 반복을 통해 보다 정확한 결과를 얻도록 노력해야 합니다. 이 과제의 초점은 세 가지 정렬 알고리즘의 시간 복잡도를 비교하는 것입니다.

제공되는 파일 목록

- profiling.pdf 본 파일
- profling.cpp 완성된 코드 & 추가 점수용 뼈대 코드
- driver.exe, driver 샘플 백만 개 정렬에 소요되는 시간 측정용 파일

Step 1. profiling.cpp 읽기

profiling.cpp 읽고 작동 방식을 숙지하세요.

profiling.exe 프로그램을 시작하는 2 가지 방법이 있습니다. 사용자는 이 파일을 실행 파일로 시작할 수 있습니다. 그러면 프롬프트에 “the number of maximum sample numbers to sort”를 입력하라는 메시지가 출력될 것입니다. 입력한 숫자가 **STARTING_SAMPLES**(sorted.h 에 저장된 magic number)보다 작으면, 적절한 메시지와 함께 프로그램을 종료합니다.

```
// sort.h
const int STARTING_SAMPLES = 1000;
```

사용자는 명령줄 인수에 샘플 수를 지정할 수 있습니다. 입력한 샘플 수가 **STARTING_SAMPLES** 보다 큰지 확인하고, 크지 않으면 적절한 메시지와 함께 프로그램을 종료합니다.

Step 2. profiling.cpp 프로그램 빌드 및 실행

- **빌드 방법:** 이전 lab 에서 본인이 작성한 **libsort.a** 및 **rand.a** 를 사용하거나 ../../lib 에 있는 해당 파일들을 사용하세요.
  ```
  $ g++ profiling.cpp -I../../include -L../../lib –lsort -lrand –o profiling,
  ```

- **실행 방법:**
  ```
  $ ./profiling.exe 30000        # PowerShell
  $  profiling 30000             # cmd
  ```

- **파일에 출력을 저장하는 방법:**
  ```
  $  ./profiling.exe 50000 > profiling.txt
  $  profiling 50000 > profiing.txt
  ```

- **스택 크기 늘리는 방법**
  이미 정렬된 입력 샘플에 수행되는 퀵 정렬은 **많은 스택 메모리**가 필요하므로 수행을 완료하지 못할 수도 있습니다. 이런 경우에는 스택의 기본 크기가 1MB 에 불과하므로 스택 크기를 늘려야 합니다. 다음 명령은 스택 크기를 16MB 로 늘립니다. **그런데,** 이 컴파일러 옵션은 Windows PowerShell 이나 Atom 콘솔에서는 **사용할 수 없습니다**. 이 명령을 실행하기 전에 PowerShell 을 **cmd windows** 로 변경해야 합니다.

```
$ cmd
$ g++ -Wl,--stack,16777216 profiling.cpp -I../../include -L../../lib –lsort –lrand –o profiling
$ ./profiling
```

또한, **mac 사용자**들은 명령어가 조금 다를 수 있으므로 구글링해보는 것을 권장합니다.
다음 명령어를 시도해볼 수 있습니다 (16M bytes): **-Wl,-stack_size,0x1000000**

Mac 사용자: (고유한 static libraries 를 가지고 있지 않은 경우 -lsort_mac 및 -lrand_mac 를 사용해보세요.)

```
$ g++ -Wl,-stack_size -Wl,1000000 profiling.cpp -I../../include -L../../lib –lsort -lrand  -o profiling
$ ./profiling
```

Step 3. 시간 복잡도 비교

Profiling.cpp 의 코드는 이미 필요에 따른 정렬 기능을 실행하고 있습니다. **삽입 정렬**, **병합 정렬**, **퀵 정렬** 이 세 가지 정렬 알고리즘의 경과 시간을 비교하려고 합니다.

1. Sorted
2. Randomly ordered
3. Reversed

실행 예시: (공간 절약을 위해 출력의 일부분만 보여줍니다.)

```
$ ./profiling 10000
The minimum number of entries is set to 1000
Enter the number of max entries to sort: 10000
The maximum sample data size is 10000
        insertionSort(): sorted
         N       repetitions       sort(sec)
      1000            219992        0.000005
      2000            121951        0.000008
      3000             90212        0.000011
      4000             70272        0.000014
      5000             56832        0.000018
      6000             48603        0.000021
      7000             40910        0.000024
      8000             35896        0.000028
      9000             32081        0.000031
     10000             28353        0.000035
        insertionSort(): randomized
         N       repetitions       sort(sec)
      1000              1299        0.000770
      2000               351        0.002855
      3000               152        0.006579
      4000                90        0.011144
      5000                69        0.014565
      6000                44        0.023295
      7000                35        0.028571
      8000                28        0.036643
      9000                21        0.047810
     10000                16        0.063062
        insertionSort(): reversed
         N       repetitions       sort(sec)
      1000               551        0.001815
      2000               202        0.004960
      3000                92        0.010967
      4000                46        0.021761
      5000                33        0.030485
      6000                23        0.043783
      7000                18        0.057889
      8000                13        0.077769
      9000                11        0.095909
     10000                 9        0.118778
quickSort(): randomized
         N       repetitions       sort(sec)
      1000              1054        0.000949
      2000               586        0.001706
      3000               361        0.002773
```

```
4000            242         0.004145
5000            210         0.004781
6000            172         0.005814
7000            146         0.006897
8000            120         0.008350
9000            107         0.009346
10000           101         0.009941
```

Step 4. 측정된 시간으로 성장률 계산

대부분의 알고리즘이 실행 시간의 대략적인 증가 기준을 가지고 있다고 가정합니다:

$$T(N) \approx a\,N^b$$

프로파일링을 통해 얻은 데이터에서 상수 "a"와 증가율 "b"를 계산하려고 합니다. a 와 b 를 알면, N 이 10 억 또는 100 만일 때, 몇 년 이상 걸릴 수도 있는 프로파일링을 실행하지 않고도 T(N)을 계산할 수 있습니다.

실행 시간을 예측하기 위해 마지막으로 계산한 실행 시간에 2^b와 2N 을 필요한 만큼 곱하세요. T(2N)과 T(N)의 비율을 이용해 b 를 계산하세요.

T(N) ≈ a N^b , T(2N) = a $(2N)^b$ 이므로, 따라서

$$\frac{T(2N)}{T(N)} = \frac{a(2N)^b}{aN^b} = \frac{2^b(N)^b}{N^b} = 2^b$$

양변에 로그를 취합니다.

$$log\frac{T(2N)}{T(N)} = \log 2^b$$

$$b = log\frac{T(2N)}{T(N)}$$

이 예시에서는 위에서 다룬 삽입 정렬의 average case 인 N = 4000 또는 2N = 8000 을 선택합니다. 여기서 사용하는 **로그의 밑(log base)은 2** 라는 점을 기억하세요.

$$b = log\frac{T(2N)}{T(N)} = log\frac{t2(8000)}{t1(4000)} = log\frac{0.036643}{0.011144} = 1.717$$

아래 표의 b 는 계산 과정을 보여줘야 합니다. 계산기를 이용해서 소수점 아래 최대 두 번째 자리까지 계산할 수 있습니다. 위 예시에서 삽입 정렬의 average case 는 1.717 이 나왔습니다. 삽입 정렬의 worst case 는 2.0 에 가깝고, 퀵 정렬의 average case 는 1.2 ~ 1.5 에 가깝습니다.

이 b = 1.717 을 사용하여 N = 4000, T(N) = 0.011144 일 때 a 를 구합니다.

$$T(N) = a\,N^{1.717}$$

$$0.011144 = a\,(4000)^{1.717}$$

$$a = \frac{0.011144}{(4000)^{1.717}}$$
$$a = 7.27\text{x}10^{-9}$$

따라서 삽입 정렬 average case 의 증가율 **b = 1.717, 상수** $a = 7.27\text{x}10^{-9}$입니다.

Step 5. 샘플 백만 개의 시간 복잡도

이 step 에서 대답하고자 하는 질문은 **"입력 크기에 대한 함수를 수행하는 내 프로그램의 수행 시간은 얼마일까?"**입니다. 이 질문에 답하기 위해 x 축에 크기 N, y 축에 실행 시간 T(N)을 두고 데이터를 그래프로 나타냅니다. 실행 시간이 입력 크기에 대한 함수라고 가정해 봅시다.

$$\text{T(N)} = a\,N^{b},$$

a 는 상수, b 는 증가율입니다. 이전 step 에서 a = 7.27 x 10^{-9} and b = 1.717 을 이미 계산했지만, 아래와 같이 N = 8000 일 때 상수 a 를 다시 계산해봅시다.

$$\text{T(8000)} = 0.036643 = \text{a}\ 8000^{\,1.717}$$

$$\text{a} = \frac{0.036643}{8000^{\,1.717}} = 7.269\,x\,10^{-9}$$

$$\text{T(N)} = 7.27\,\text{x}10^{-9}\ \text{N}^{\,1.717}$$

결론적으로, 두 상수 a 가 같다는 것을 확인했습니다. 이제 샘플 백만 개 또는 십억 개의 경과 시간을 추정할 수 있습니다. 이 a 와 b 를 사용하면 프로그램을 실행하지 않고도 N = 1,000,000 또는 샘플이 십억 개일 때 T(N)의 estimated time 을 계산할 수 있습니다, 그렇죠? ^^

본인의 컴퓨터로 계산한 **증가율 b 와 상수 a** 로 샘플 백만 개의 estimated time 을 **계산**하세요. 보고서에 estimated time 을 계산하는 정확한 과정을 작성하세요. 더 많은 샘플로 계산한 상수 "a"를 사용하세요. **적절한 시간 단위를 사용하세요. 예를 들어, 1,234.57 초가 아닌, 20 분 35 초라고 작성하세요.**

또한, 본인의 컴퓨터로 100 만 개의 샘플을 실행하여 estimated 와 measured 경과 시간을 빈칸에 작성하세요. **이 용도로 driver.exe 를 사용하세요.**

	$\text{T(N)} \approx a\,N^{b}$, a = b = insertionsort - Best		$\text{T(N)} \approx a\,N^{b}$, a = b = insertionsort - Average		$\text{T(N)} \approx a\,N^{b}$, a = b = insertionsort - Worst	
N	10,000	Time for Million	10,000	Time for Million	10,000	Time for Million
Time		Estimated:		Estimated:		Estimated:
N	20,000		20,000		20,000	
Time		Measured:		Measured:		Measured:

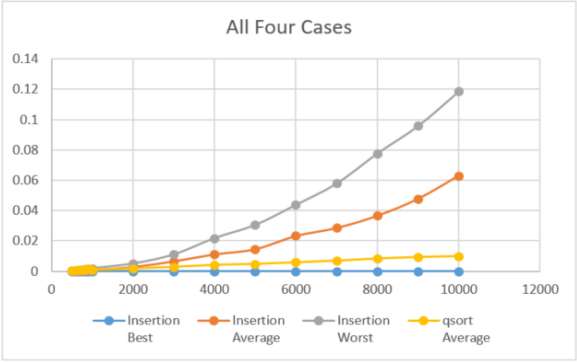
	$T(N) \approx a\,N^{b}$, a = b = Average quicksort **O(N log N): randomized**	
N	10,000	Time for Million
Time		Estimated:
N	20,000	
Time		Measured:

	$T(N) \approx a\,N^{b}$, a = b = Average mergesort **O(N log N): randomized**	
N	10,000	Time for Million
Time		Estimated:
N	20,000	
Time		Measured:

Step 6. 시간 복잡도 비교 그래프

이전 step 에서 구한 데이터 집합을 아래와 같이 도표로 비교하기 위해 **그래프를 그리세요**. Excel Chart(분산형)을 이용하여 그래프를 그리면 됩니다. 삽입 정렬과 퀵 정렬의 데이터를 합친 그래프와 보고서 예시입니다. **보고서의 그래프와 표에 병합 정렬의 average case 도 반드시 포함해야 합니다.**

n	Insertion Best	Insertion Average	Insertion Worst	qsort Average
500	0.000003	0.000180	0.000341	0.000380
600	0.000003	0.000256	0.000446	0.000657
700	0.000004	0.000374	0.000667	0.000652
800	0.000004	0.000477	0.001075	0.000716
900	0.000004	0.000623	0.000990	0.001653
1000	0.000005	0.000770	0.001815	0.000949
2000	0.000008	0.002855	0.004960	0.001706
3000	0.000011	0.006579	0.010967	0.002773
4000	0.000014	0.011144	0.021761	0.004145
5000	0.000018	0.014565	0.030485	0.004781
6000	0.000021	0.023295	0.043783	0.005814
7000	0.000024	0.028571	0.057889	0.006897
8000	0.000028	0.036643	0.077769	0.008350
9000	0.000031	0.047810	0.095909	0.009346
10000	0.000035	0.063062	0.118778	0.009941



Step 7. profiling.cpp 에 DRY 적용하기

profiling.cpp 를 **profile.cpp** 에 복사합니다. 이 step 을 구현한 후 **profile.cpp** 를 제출하세요. The final output of **profile.cpp** 의 최종 출력은 **profiling.cpp** 의 출력과 같아야 합니다.

Profile.cpp 에 3 번 반복되는 코드가 있습니다. 이는 우리의 코딩 원칙 중 하나인 DRY-Do not repeat yourself 에 어긋납니다. 보다시피 세 부분의 유일한 차이점은 정렬 함수와 출력문입니다. 더 많은 정렬 함수를 테스트할 때, 동일한 코드를 여러 번 반복하지 않기 위해 정렬 함수들과 그 종류를 문자열 배열에 넣으려고 합니다. 예를 들어, 정렬의 종류를 다음과 같이 저장할 수 있습니다:

```
vector<string> sort_st = {"insertionsort", "mergesort", "quicksort"};
```

그렇다면 정렬 함수들은 어떻게 저장할 수 있을까요?

전에 배운 함수 포인터를 기억하시나요? 삽입 정렬은 다음과 같이 함수 포인터 sort_fp 를 사용하여 정의할 수 있습니다:

```
void (*sort_fp)(int*, int, bool (*comp)(int, int)) = insertionsort;
```

함수 포인터의 배열을 선언하는 방법은 다음과 같습니다:

```
void (*sort_fp[])(int*, int, bool (*comp)(int, int));
```

이제 세 가지 정렬 알고리즘의 배열을 초기화하는 방법과 profiling.cpp 의 나머지 부분을 DRY 와 NMN 원칙에 어긋나지 않게 코딩할 수 있습니다.

과제 제출

- **On my honour, I pledge that I have neither received nor provided improper assistance in the completion of this assignment.**
  서명: _____ 학번: _____
- 제출하기 전에 코드가 제대로 컴파일이 되고 실행되는지 확인하세요. 제출 직전에 급하게 코드를 수정한 후 코드가 제대로 컴파일이 될 거라고 짐작하지 않는 게 좋습니다. “거의” 작동하는 코드도 틀린 것입니다.
- 과제가 컴파일 및 실행된다면, 마감 기한 전까지 과제의 일부만 완성했더라도 제출하기 바랍니다. 마감 시간 이후 24 시간 이내 제출하면, 만점에서 25% 감점하고 채점합니다. 그 이상 늦은 것은 채점하지 않으며, 0 점 처리합니다.
- 제출 후, **마감 기한 전까지** 수정 및 재제출이 가능합니다. 파일 하나만 수정하더라도 해당 파일과 관련된 파일들을 모두 재제출해야 합니다. 재제출 횟수는 제한 없습니다. 마감 기한 전에 **가장 마지막으로** 제출된 파일을 채점할 것입니다.

제출 파일 목록

다음 파일들을 Piazza 의 pset 폴더에 제출하세요.

- **ProfilingReport.docx** (docx 또는 읽을 수 있는 형식), 보고서는 다음 내용을 포함해야 합니다:
  1. profiling.exe 의 출력 스크린 캡처본
  2. 본인의 데이터(estimated 와 measured)를 사용해서 완성한 성능 분석 표
     a, b, estimated time 계산 과정을 **5 가지 케이스**(삽입 정렬의 worst, average, best case, 그리고 병합 정렬과 퀵 정렬의 average case) 모두 작성하세요.
  3. **5 가지 케이스**를 비교하기 위한 엑셀 차트와 그래프
  4. 5 가지 알고리즘 케이스의 시간 복잡도를 관찰하고 그에 대한 **본인의 의견 서술.**
- **profile.cpp**
  1. DRY 와 NMN 원칙이 모두 적용되어야 합니다.

마감 기한 & 배점

- 마감 기한: 11:55 pm