INHA UNIVERSITY

알고리즘설계 HW #1

Sorting

보고서 작성 서약서

1. 나는 타학생의 보고서를 베끼거나 여러 보고서의 내용을 짜집기하지 않겠습니다.
2. 나는 보고서의 주요 내용을 인터넷사이트 등을 통해 얻지 않겠습니다.
3. 나는 보고서의 내용을 조작하지 않겠습니다.
4. 나는 보고서 작성에 참고한 문헌의 출처를 밝히겠습니다.
5. 나는 나의 보고서를 제출 전에 타학생에게 보여주지 않겠습니다.

나는 보고서 작성시 윤리에 어긋난 행동을 하지 않고 정보통신공학인으로서 나의 명예를 지킬 것을 맹세합니다.

2020 년 5 월 1 일

학부 정보통신공학과
학년 4
성명 이정우
학번 12171833

최소한 아래의 내용을 포함할 것.

1. 개요

* 아래 네 가지 정렬 알고리즘을 구현하고 성능을 비교 분석하라.(100점)

(1) Selection sort
(2) Median-of-three Quick sort
(3) Shell Sort
(4) Bitonic sort
(5) Odd-Even Merge sort

- 데이터 셋의 크기(n)를 변화시켜가면서(예를 들어 n=10,000일 때, n=500,000일 때, n=1,000,000일 때, n=5,000,000일때) 데이타셋을 random하게 생성할 것.
- 위의 n는 예를 든 것뿐이며 재량껏 늘려서 실험 할 것.
- 동일한 데이타셋을 다섯개 알고리즘의 입력으로 주었을 때 실행시간을 측정할 것.
- n에 대한 성능 비교 그래프를 제시할 것.
- 각 알고리즘의 성능에 대해 자세히 비교 분석을 할 것.

2. 상세 설계내용

(1) Selection sort

```
void selection_sort(int *arr, int length){
    for(int i=0; i<length-1; i++){
        int min=i;
        for(int j=i; j<length; j++){
            if(arr[j] < arr[min]) min=j;
        }
        if(min != i) swap(arr[i], arr[min]);
    }
}
```

(2) Median-of-three Quick sort

```
void mot_quick_process(int *arr, int s, int e){
    int bs=s, be=e;
    int m = (s+e)/2;
    if(arr[s]>arr[m]) swap(arr[s], arr[m]);
    if(arr[m]>arr[e]) swap(arr[m], arr[e]);
    if(arr[s]>arr[m]) swap(arr[s], arr[m]);
    if(e-s<3) return ;

    int pivot = arr[m];
    swap(arr[bs+1], arr[m]);
    s++; e--;
    //12171833 이정우
    while(s<e){
        while(pivot<=arr[e] && s<e) e--;
        if(s>e) break;
        while(pivot>=arr[s] && s<e) s++;
        if(s>e) break;
        swap(arr[s], arr[e]);
    }
    swap(arr[bs+1], arr[s]);
    if(bs<s) mot_quick_process(arr, bs, s-1);
    if(be>e) mot_quick_process(arr, s+1, be);
}
```

맨 처음 array 의 처음, 중간, 끝 값을 비교하여 중간 값을 pivot 으로 사용한다.
처음과 끝 값 사이의 차이가 3 보다 작을 경우 pivot 을 찾는 과정에서 정렬의 완료되기 때문에 return 해준다.
중간 값의 위치를 처음+1 의 위치로 옮기고 2 개의 값은 이미 비교를 하였기 때문에 처음+1, 끝-1 부터 pivot 과 비교를 하여 값을 swap 해준다.
마지막에 pivot 값을 배열의 pivot 보다 작은 값들의 맨 끝으로 옮기고 pivot 의 좌, 우 배열을 quick sort 하는 것을 반복한다.

```
void mot_quick_sort(int *arr, int length){
    mot_quick_process(arr, 0, length-1);
}
```

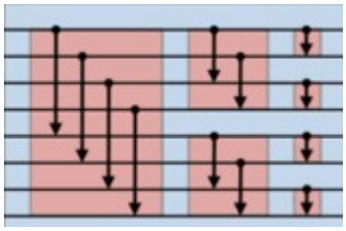
(3) Shell sort

```
void shell_sort(int *arr, int length){
    int h=1;
    while(h<length-1) h=h=3*h+1;
    h/=3;
    //12171833 이정우
    while(h){
        for (int i=h; i<=length; i++){
            int v = arr[i];
            int j = i;
            while (j>=h && arr[j-h]>v){
                arr[j] = arr[j-h];
                j = j-h;
            }
            arr[j] = v;
        }
        h/=3;
    }
}
```

맨 처음 h 의 크기를 찾은 후 h 의 크기가 1 의 될 때까지 3 으로 나누어 주며
h 의 간격대로 insertion sort 를 수행한다.

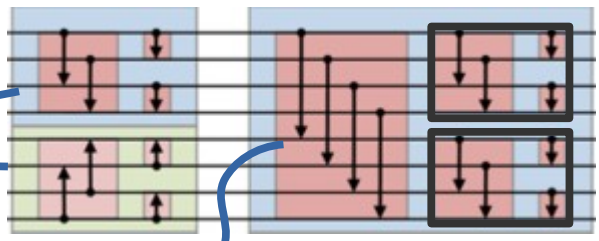
(4) Bitonic sort

```
void bitonic_1d(bool up_flag, int *arr, int s, int e){
    if(e-s==1) return ;
    int d = (e-s)/2;
    int m = (s+e)/2;
    for(int i=s; i<m; i++){
        if ((arr[i]>arr[i+d]) == up_flag)
            swap(arr[i], arr[i+d]);
    }   //12171833 이정우
    bitonic_1d(up_flag, arr, s, m);
    bitonic_1d(up_flag, arr, m, e);
}
```



배열의 s~e-1 사이의 값을 같은 방향으로 array 의 크기를 반으로 나누며 반복 정렬한다.

```
void bitonic_2d(bool up_flag, int *arr, int s, int e){
    if(e-s==1) return ;     //12171833 이정우
    int d = (e-s)/2;
    int m = (s+e)/2;
    bitonic_2d(true, arr, s, m);
    bitonic_2d(false, arr, m, e);
    for(int i=s; i<m; i++){
        if ((arr[i]>arr[i+d]) == up_flag)
            swap(arr[i], arr[i+d]);
    }
    bitonic_1d(up_flag, arr, s, m);
    bitonic_1d(up_flag, arr, m, e);
}
```



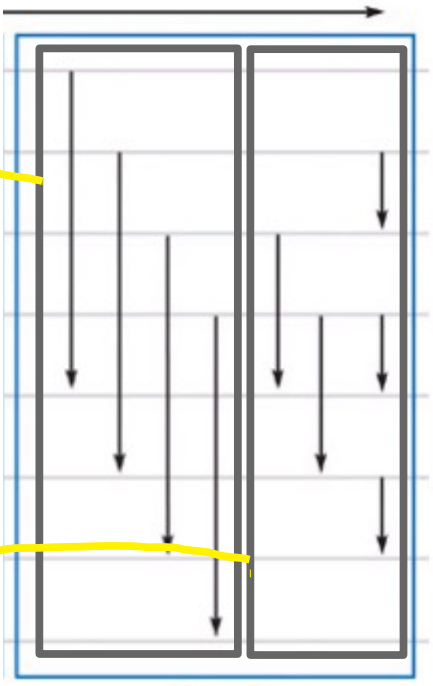
처음의 배열을 각기 다른 방향으로 반 씩
나누어 bitonic_2d 를 불러준다.

```
void bitonic_sort(int *arr, int length){
    bitonic_2d(true, arr, 0, length);
}
```

다음으로 자기 자신을 up_flag 방향으로 정렬하고, 그 bitonic_1d 를 불러주어 반으로 나누어가며 up_flag 방향으로 계속 정렬하도록 한다.

(5) Odd-Even Merge sort

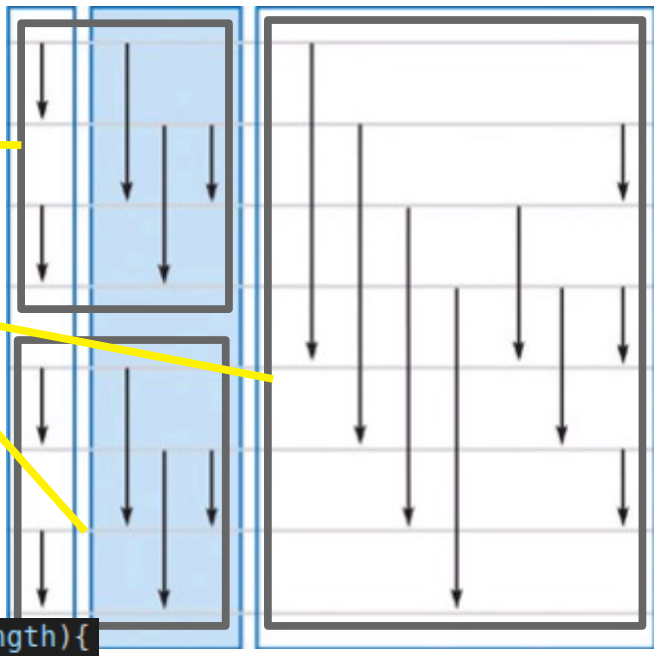
```
void oem_swap(int *arr, int s, int e){
    int d = (e-s)/2;
    int m = (s+e)/2;
    for(int i=s; i<m; i++){
        if (arr[i]>arr[i+d])
            swap(arr[i], arr[i+d]);
    }       //12171833 이정우
    while(true){
        d/=2;
        if(d==0) break;
        int i=s+d;
        while(d<e-d-i){
            if (arr[i]>arr[i+d])
                swap(arr[i], arr[i+d]);
            i++;
        }
    }
}
```



처음에 (e-s)/2 의 간격 만큼으로 s 부터 e 까지 정렬을 한 후

d 를 2 로 나눠가면서 s+d 부터 s-d 까지 d 간격으로 정렬을 d 가 0 보다 큰 동안 반복한다.

```
void oem_process(int *arr, int s, int e){
    if(e-s==1) return ;
    int m = (s+e)/2;    //12171833 이정우
    oem_process(arr, s, m);
    oem_process(arr, m, e);
    oem_swap(arr, s, e);
}
```



```
void odd_even_merge_sort(int *arr, int length){
    oem_process(arr, 0, length);
}
```

3. 실행 화면

```
double test_sorting(void (*sort)(int *, int), int *arr, int n){
    int *tmp = new int[n];
    for(int i=0; i<n; i++) tmp[i] = arr[i];
    clock_t s, e;
    s = clock();
    sort(tmp, n);
    e = clock();
    return (double)(e-s)/CLOCKS_PER_SEC;
}
```

(1), (2), (3)번 sorting 비교

Sort / Data number	**1,000**	**10,000**	**100,000**	**1,000,000**
Selection sort	0.00366 (s)	0.163801 (s)	13.2164 (s)	
Median-of-three Quick sort	0.000189 (s)	0.001344 (s)	0.018529 (s)	
Shell sort	0.000239 (s)	0.001926 (s)	0.037895 (s)	

4. 분석 및 결론

프로그램을 코딩하면서 처음에 설계했던 UML 대로만으로는 개발되지 않았다.

UML 을 짰을 당시에는 다형성을 고려하지 못해 OpenedSNS 와 ClosedSNS 에 SNS 에 없는 변수를 만들어 사용하도록 설계 했었는데 실제로 프로그램을 실행해보니 그렇게 하면 다형성을 활용하지 못함을 알고 코드를 수정하였다.

또한, 처음에 header.h 라는 파일에 모든 파일의 헤더를 정의 해놓고, 모든 파일에 header.h.만을 include 하면 된다고 생각하였었는데, 그렇게 하니 헤더파일이 꼬여서 컴파일 하였을 때 에러가 발생하였으나, 헤더파일을 따로 include 해주고 난 뒤에 해결되었다.

string 클래스를 제대로 활용해 본 것은 이번이 처음인데, 수업시간에 string 의 경우 문자열이 heap 메모리 영역에 저장된다고 한 것이 기억나 한번 실제로 확인해보았다.

```
#include <iostream>
#include <string>
#include <cstdlib>
using namespace std;

int main(void){
    string a="asd";
    string b="asdasd";
    cout<<sizeof(a)<<endl<<sizeof(b);
    return 0;
}
```

```
C:₩Users₩jng-u₩Desktop₩jung-u₩develop>g++ -o test test.cpp

C:₩Users₩jng-u₩Desktop₩jung-u₩develop>test
24
24
```

이것을 통해 stirng 클래스에서 문자열이 heap 메모리에 저장된다는 것을 알 수 있었다.

```
#include <iostream>
#include <string>
#include <cstdlib>
using namespace std;

int main(void){
    string a="asd";
    string *b = new string("asdasd");
    cout<<sizeof(a)<<endl<<sizeof(b);
    return 0;
}
```

```
C:₩Users₩jng-u₩Desktop₩jung-u₩develop>g++ -o test test.cpp

C:₩Users₩jng-u₩Desktop₩jung-u₩develop>test
24
4
```

그러나 string 클래스 자체가 24bytes 를 차지하기 때문에 프로그램을 짜면서 string 객체는 모두 포인터를 이용하여 선언하였다.

클래스를 상속할 때 protected 를 이용하지 않고 private 과 scope operator 를 사용하여 코딩하였다.

크기가 큰 객체의 경우는 될 수 있다면 포인터를 이용하여 stack memory 사용을 줄였다.

코딩을 하면서 파일의 개수가 많아지니 부모클래스에서의 사소한 것이 바뀌면 자식클래스에까지 영향을

끼쳐 수정하는데 굉장히 번거로웠다. 클래스를 처음 설계할 때 잘 설계하는 것이 중요함을 느꼈다.