

# Deap

학번 : 201002513

이름 : 최 혁수

## ☐ 실습 7. Deap

- 개요 & 알고리즘
- 주요 소스코드
- 실행결과 분석 & 구현상의 오류 및 한계

## ☐ 소감문 & 한계성

## ☐ All Source Code

# 실습 7. Deap

- 개요 & 알고리즘
- 주요 소스코드
- 실행결과 분석 & 구현상의 오류 및 한계

# Deap

## □ 개요

대칭 최소-최대 힙(Symmetric Min\_Max Heap)을 구현하여 MinHeap, MaxHeap 재구현하며 기존 HeapSort와 비교하여 시간적 우위를 알아본다.

## □ 접근방법

- 맨 마지막 Array에 대한 index로 Max or MinHeap에 속해있는지 확인 후 Partner를 구해 SWAP을 구현한다.
- 기존 HeapSort를 사용하여 Deap 과의 시간적 우위를 계산한다.
- Partner : Min/MaxHeap과의 Index 비교를 통해 파트너를 구한다.

## □ Deap Algorithm

Index=1을 Root로 기준하여 왼쪽 자식(=MinHeap), 오른쪽 자식(=MaxHeap)을 구현하여 서로 다른 대칭 힙을 가진다.

# Deap(1/3)

## □ head.h

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#define MAX_SIZE 128 // MAX_SIZE 정의
#define SWAP(x,y,t) ((t)=(x),(x)=(y),(y)=(t))
#define TRUE 1 // boolean 함수 대체
#define FALSE 0

int minPartner(int n);
int maxPartner(int n);
int WhereHeap(int n); // deapMaxSize가 Min or Maxheap 어느위치에 있는지 확인
void Insert(int item);
void adjustDeap(int n); // heap 재조정
int deleteMin();
int deleteMax();
void print();
void printSortedOrder(); // heap sort -> deap sort 순으로 print
```

## □ minPartner(int), maxPartner(int)

```
int minPartner(int n){ // find the partner
    int h;
    for(h=2; 2*h<=n; h*=2); // sibling 갯수를 count
    return n-h/2; // deapMaxSize - sibling/2
}

int maxPartner(int n){ // find the partner
    int h;
    for(h=2; 2*h<=n; h*=2);
    if (n+h/2<=deapMaxSize ) return n+h/2; // partner이 존재할 경우 리턴
    else return (n+h/2)/2; // 없으면 부모가 리턴
}
```

# Deap(2/3)

## □ void Insert(int)

```
void Insert(int item){
    int temp;
    int partner;
    deapMaxSize++; // Insert 마다 Size증가 = 쓰일 배열의 크기 증가
    deap[deapMaxSize] = item;
    if(WhereHeap(deapMaxSize)){ // MaxHeap에 있는 경우
        partner = minPartner(deapMaxSize); //minheap에 있는 partner 확인
        if(deap[deapMaxSize] < deap[partner]){ // 파트너와 비교해서 현재 값이 작으면 SWAP
            SWAP(deap[deapMaxSize], deap[partner], temp);
            adjustDeap(partner); // Adjust deap
        }
    }
    else{ // Maxheap에 있는 경우
        partner = maxPartner(deapMaxSize);
        // partner이 Root가 아닌 경우 및 파트너보다 큰 경우
        if(partner!=1 && deap[deapMaxSize] > deap[partner]){
            SWAP(deap[deapMaxSize], deap[partner], temp);
            adjustDeap(partner);
        }
    }
    // 비교문이 끝나고 현재 deapMaxSize가 있는 위치의 deap을 재구성한다.
    adjustDeap(deapMaxSize);
}
```

## □ void adjustDeap(int)

```
void adjustDeap(int n){
    int i;
    int temp;
    if(WhereHeap(n)){ // MaxHeap에 있는 경우
        for(i=n; i>3; i/=2) // MaxHeap의 Root까지 반복하면서 부모값이 자식보다 작을 경우
            if(deap[n] > deap[n/2]) SWAP(deap[n], deap[n/2], temp);
    }
    else{
        for(i=n; i>2; i/=2)
            if(deap[n] < deap[n/2]) SWAP(deap[n], deap[n/2], temp);
    }
}
```

# Deap(3/3)

## ❑ void Insert(int)

```
void printSortedOrder(){ // Heap, Deap Sort 출력
    int i, j;
    int p, temp, heap[128];
    clock_t start, end;

    for(i=0; i<=deapMaxSize; i++) heap[i] = deap[i+1]; // Deap->Heap 복사

    printf("===PRINT Sorted Order===\n");
    printf("[Using HeapSort]\n");

    start = clock(); // heap에 대한 start clock

    for(i=2; i<=deapMaxSize-1; i++)
    for(j=i; heap[j/2]<heap[j] && j>1; j/=2) SWAP(heap[j/2], heap[j], temp);
    //부모와 자식값을 비교하면서 MAX_Heap 생성

    for(i=1; i<deapMaxSize-1; i++){
        SWAP(heap[1], heap[deapMaxSize-i], temp); // Root <-> deapMaxSize-1을 변경
        // (deapMaxSize-1)-i 이후는 정렬되었으므로, 이전까지 정렬
        for(j=1; j*2<deapMaxSize-1-i; )
            if(j*2+1<=deapMaxSize-1-i) // 왼쪽, 오른쪽 자식이 모두 존재할때
                if(heap[j*2] > heap[j*2+1]) // 큰 값을 찾아 다시 자신과 비교하여 SWAP
                    if(heap[j*2] > heap[j]) SWAP(heap[j], heap[j*2], temp), j*=2;
                else break;
            else if(heap[j*2+1] > heap[j]) SWAP(heap[j], heap[j*2+1], temp), j=j*2+1;
            else break;
            else // 왼쪽자식만 있을때
                if(heap[j*2] > heap[j]) SWAP(heap[j], heap[j*2], temp), j*=2;
    }

    for(i=1; i<deapMaxSize; i++) printf("%3d\n", heap[i]);
    end = clock();

    printf("Duration : %.3f\n", (double)(end-start)/CLK_TCK);
    printf("[Using Deap's Deletion]\n");

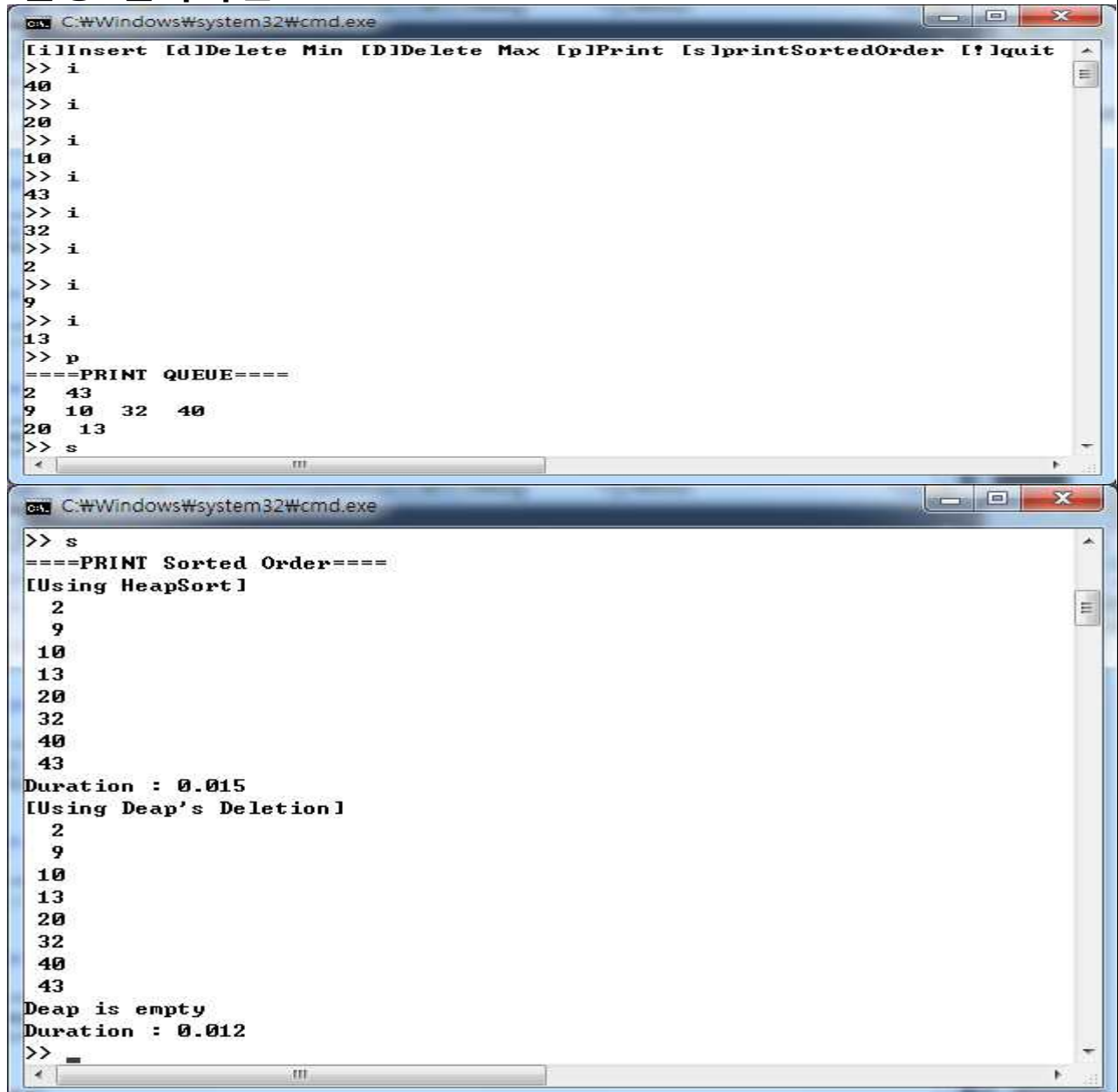
    p=deapMaxSize; // deleteMin()에서 deapMaxSize를 -1 하므로 p에 저장해서 사용

    start = clock(); // deap에 대한 start clock
    for(i=2; i<=p; i++){
        printf("%3d\n", deleteMin());
    }
    end = clock();

    printf("Deap is empty\n");
    printf("Duration : %.3f\n", (double)(end-start)/CLK_TCK);
}
```

# 실행결과 분석 & 구현상의 오류 및 한계

## □ 실행 결과화면



```
C:\Windows\system32\cmd.exe
[il]Insert [d]Delete Min [D]Delete Max [p]Print [s]printSortedOrder [!]quit
>> i
40
>> i
20
>> i
10
>> i
43
>> i
32
>> i
2
>> i
9
>> i
13
>> p
====PRINT QUEUE====
2 43
9 10 32 40
20 13
>> s

C:\Windows\system32\cmd.exe
>> s
====PRINT Sorted Order====
[Using HeapSort]
2
9
10
13
20
32
40
43
Duration : 0.015
[Using Deap's Deletion]
2
9
10
13
20
32
40
43
Deap is empty
Duration : 0.012
>>
```

## □ 구현상의 오류 및 한계

deap의 Max\_SIZE를 정의하고 있어 그 이상의 insert는 할당된 영역을 넘어가므로 error가 발생한다. 또한 위 HeapSort/DeapSort 의 시간적 우위를 계산할 때 Duration이 소수점이하로 나오므로 정확한 시간적 차이를 계산하기는 어렵다.



# 소감문 & 한계성

# 소감문 & 한계성

## □ 소감문

본 Deap 구현을 통해 MaxHeap / MinHeap을 대칭적으로 한 트리에 속하여 표현이 가능한 것을 알았으며 실습 PDF와는 다르게 Deap을 구현하여 Sort하는 것이 일반 HeapSort보다 시간적으로 절약된다는 것을 알수 있었다.

## □ 한계성

deap의 크기할당 문제와 시간계산에 대한 정확한 차이를 비교하기 어렵다는 점 및 배열로 표현되었기 때문에 시각적인 설명이 어려워 코드상에 문제가 생겼을 때 이를 처음부터 끝까지 컴파일 하는 형식으로 찾아야 한다는 시각적인 표현문제 또한 한계성을 지니고 있다.

**All Source Code**

# Deap(1/4)

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#define MAX_SIZE 128 // MAX_SIZE 정의
#define SWAP(x,y,t) ((t)=(x),(x)=(y),(y)=(t))
#define TRUE 1 // boolean 함수 대체
#define FALSE 0

int minPartner(int n);
int maxPartner(int n);
int WhereHeap(int n); // deapMaxSize가 Min or Maxheap 어느위치에 있는지 확인
void Insert(int item);
void adjustDeap(int n); // heap 재조정
int deleteMin();
int deleteMax();
void print();
void printSortedOrder(); // heap sort -> deap sort 순으로 print

int deap[MAX_SIZE]; // Array of Deap
int deapMaxSize = 1; // Deap Size

int WhereHeap(int n){ // deapMaxSize가 Max or MinHeap 중 어디에 있는지 확인
    int h;
    for(h=2; 2*h<=n; h*=2); // sibling 갯수를 count
    if(n-h > h/2-1) return TRUE; // 절반이상? TRUE : FALSE
    else return FALSE;
}

int minPartner(int n){ // find the partner
    int h;
    for(h=2; 2*h<=n; h*=2); // sibling 갯수를 count
    return n-h/2; // deapMaxSize - sibling/2
}

int maxPartner(int n){ // find the partner
    int h;
    for(h=2; 2*h<=n; h*=2);
    if (n+h/2<=deapMaxSize ) return n+h/2; // partner이 존재할 경우 리턴
    else return (n+h/2)/2; // 없으면 부모가 리턴
}

void Insert(int item){
    int temp;
    int partner;
    deapMaxSize++; // Insert 마다 Size증가 = 쓰일 배열의 크기 증가
    deap[deapMaxSize] = item;
    if(WhereHeap(deapMaxSize)){ // MaxHeap에 있는 경우
        partner = minPartner(deapMaxSize); //minheap에 있는 partner 확인
        if(deap[deapMaxSize] < deap[partner]){ // 파트너와 비교해서 현재 값이 작으면 SWAP
            SWAP(deap[deapMaxSize], deap[partner], temp);
            adjustDeap(partner); // Adjust deap
        }
    }
    else{ // Maxheap에 있는 경우
        partner = maxPartner(deapMaxSize);
        // partner이 Root가 아닌 경우 및 파트너보다 큰 경우
        if(partner!=1 && deap[deapMaxSize] > deap[partner]){
            SWAP(deap[deapMaxSize], deap[partner], temp);
            adjustDeap(partner);
        }
    }
    // 비교문이 끝나고 현재 deapMaxSize가 있는 위치의 deap을 재구성한다.
    adjustDeap(deapMaxSize);
}
```

# Deap(2/4)

```
void adjustDeap(int n){
    int i;
    int temp;
    if(WhereHeap(n)){ // MaxHeap에 있는 경우
        for(i=n; i>3; i/=2) // MaxHeap의 Root까지 반복하면서 부모값이 자식보다 작을 경우
            if(deap[n] > deap[n/2]) SWAP(deap[n], deap[n/2], temp);
    }
    else{
        for(i=n; i>2; i/=2)
            if(deap[n] < deap[n/2]) SWAP(deap[n], deap[n/2], temp);
    }
}

int deleteMin(){
    int i, temp;
    if(deapMaxSize<2) printf("Error!!\n"), exit(0);
    temp=deap[2]; // MinHeap의 Root를 임시저장

    for(i=2; 2*i<=deapMaxSize; ) // deapMaxSize까지 반복
        if(2*i+1<=deapMaxSize){ // 오른 자식이 있을 때
            if(deap[i*2]<deap[i*2+1]){ // 두 자식값을 비교하여 작은값을 부모값에 저장
                deap[i] = deap[i*2], i=i*2;
            }
            else{
                deap[i] = deap[i*2+1], i=i*2+1;
            }
        }
        else{ // 왼쪽 자식만 있을 때
            deap[i] = deap[i*2], i=i*2;
        }
    }
    deap[i] = deap[deapMaxSize--]; // deapMaxSize를 줄인다.
    return temp;
}

int deleteMax(){ // deleteMin()과 동일
    int i, temp;
    if(deapMaxSize<2) printf("Error!!\n"), exit(0);
    else if(deapMaxSize==2) deleteMin(); // deapMaxSize==2는 MinHeap의 Root idx이다.
    else{
        temp = deap[3]; // MaxHeap의 Root의 idx는 3이다.
        for(i=3; 2*i<=deapMaxSize; )
            if ( 2*i+1 <= deapMaxSize )
                if(deap[i*2] < deap[i*2+1]) deap[i]=deap[i*2+1], i=i*2+1;
                else deap[i]=deap[i*2], i=i*2;
            else{
                deap[i]=deap[i*2], i=i*2;
            }
        }
        deap[i] = deap[deapMaxSize--];
    }
    return temp;
}

void print(){
    int i, j;
    printf("====PRINT QUEUE====\n");
    // Tree의 level의 sibling의 갯수만큼 한줄씩 출력
    for(i=2, j=4; i<=deapMaxSize; i++){
        if(i==j) printf("\n"), j*=2; // sibling 출력에 대한 레벨증가 및 sibling*2
        printf("%d ", deap[i]);
    }
}
```

# Deap(3/4)

```
void printSortedOrder(){ // Heap, Deap Sort 출력
    int i, j;
    int p, temp, heap[128];
    clock_t start, end;

    for(i=0; i<=deapMaxSize; i++) heap[i] = deap[i+1]; // Deap->Heap 복사

    printf("===PRINT Sorted Order===\n");
    printf("[Using HeapSort]\n");

    start = clock(); // heap에 대한 start clock

    for(i=2; i<=deapMaxSize-1; i++)
    for(j=i; heap[j/2]<heap[j] && j>1; j/=2) SWAP(heap[j/2], heap[j], temp);
    //부모와 자식값을 비교하면서 MAX_Heap 생성

    for(i=1; i<deapMaxSize-1; i++){
        SWAP(heap[1], heap[deapMaxSize-i], temp); // Root <-> deapMaxSize-1을 변경
        for(j=1; j*2<deapMaxSize-1-i; ) // (deapMaxSize-1)-i 이후는 정렬되었으므로, 이전까지 정렬
            if(j*2+1<=deapMaxSize-1-i) // 왼쪽, 오른쪽 자식이 모두 존재할때
                if(heap[j*2] > heap[j*2+1]) // 큰 값을 찾아 다시 자신과 비교하여 SWAP
                    if(heap[j*2] > heap[j]) SWAP(heap[j], heap[j*2], temp), j*=2;
                    else break;
                else
                    if(heap[j*2+1] > heap[j]) SWAP(heap[j], heap[j*2+1], temp), j=j*2+1;
                    else break;
            else // 왼쪽자식만 있을때
                if(heap[j*2] > heap[j]) SWAP(heap[j], heap[j*2], temp), j*=2;
    }

    for(i=1; i<deapMaxSize; i++) printf("%3d\n", heap[i]);
    end = clock();

    printf("Duration : %.3f\n", (double)(end-start)/CLK_TCK);
    printf("[Using Deap's Deletion]\n");

    p=deapMaxSize; // deleteMin()에서 deapMaxSize를 -1 하므로 p에 저장해서 사용

    start = clock(); // deap에 대한 start clock
    for(i=2; i<=p; i++){
        printf("%3d\n", deleteMin());
    }
    end = clock();

    printf("Deap is empty\n");
    printf("Duration : %.3f\n", (double)(end-start)/CLK_TCK);
}
```

# Deap(4/4)

```
#include "head.h"

int main(void)
{
    int temp;
    char select;

    printf("[i]Insert [d]Delete Min [D]Delete Max [p]Print [s]printSortedOrder [!]quit\n");
    while(1){
        printf(">> ");
        scanf("%c", &select);
        switch(select){
            case 'i' :
                scanf("%d", &temp);
                Insert(temp);
                break;
            case 'd' :
                printf("%5d\n", deleteMin());
                break;
            case 'D' :
                printf("%5d\n", deleteMax());
                break;
            case 'p' :
                print();
                printf("\n");
                break;
            case 's' :
                printSortedOrder();
                break;
            case '!':
                return 0;
        }
        fflush(stdin);
    }
}
```