Nonlinear Data Structure

http://smartlead.hallym.ac.kr

Instructor: **Jin Kim**
010-6267-8189(033-248-2318)

jinkim@hallym.ac.kr
Office Hours:
Office : 자연대 7618

Lab(Sorting)

http://smartlead.hallym.ac.kr

Instructor: **Jin Kim**
010-6267-8189(033-248-2318)

jinkim@hallym.ac.kr
Office Hours:

Lab : Sorting(2)

Sorting(**n log(n)**)

1. 퀵정렬 알고리즘을 구현하라(quickSort) //빨간코드사용
   이때 입력데이터는 랜덤
2. 합병정렬 알고리즘을 구현하라.(mergeSort)
3. 히프정렬 알고리즘을 구현하라.(heapSort)

* 위의 정렬 알고리즘을 사용하여 1) 랜덤 2) 정렬 3) 역정렬된 데이터에 적용하고 그래프를 만들어라(그래프1-랜덤, 그래프2-정렬, 그래프3-역정렬)

4. 지난주에 얻은 (선택, 삽입,버블) 결과와 이번주의 그래프1-랜덤을 한 그래프로 만들어라.(그래프4-통합)

이때 데이터는 1만,2만,3만,4만,…10만개의 데이터를 사용하라(입력데이터는 0에서 10만미만의 정수). X축은 데이터, y축은 시간으로 하는 2차원표를 엑셀을 사용하여 구현하라

퀵 정렬 (5)

- partition 알고리즘

```
partiton(a[], i, j) //=> public static int partition(int[] a, int i, int j){
   //middle ← (i + j) / 2;     // middle은 a[]의 중앙 인덱스 값
   // pivot ← a[middle];     // a[]의 중앙 원소값을 pivot으로 설정
   // a[middle] ← a[i];     // a[i]와 a[middle]을 서로 교환
   // a[i] ← pivot;     //a[i]는 pivot 값을 보관  제일왼쪽
   pivot=a[i];  //ppt의 알고리즘(빨간코드)
   p ← i;     // p는 두 파티션의 경계를 지시하는 인덱스
   for (k ←i+1;  k ≤ j;  k ← k+1) do  {
                        // a[i]를 제외한 a[i+1 : j]에 있는 모든 원소 a[k]들을 검사하여
      if (a[k] < pivot) then  {     // a[k]가 pivot보다 작으면
         p ← p+1;                    // p를 1 증가시켜 a[k]를 p 인덱스 범위
         temp ← a[p];               // 안으로 포함되게 함
         a[p] ← a[k];
         a[k] ← temp;
      }
   }
   temp ← a[i];     // a[i]와 a[p]를 교환
   a[i] ← a[p];
   a[p] ← temp;
   return p;
end partition()
```

퀵 정렬 (9)

- Sorting 클래스의 메소드 멤버 구현

```
public static void quickSort(int[] a)  {
// 퀵 정렬의 메인 메소드
    internalQuickSort(a, 0, a.length-1);
}

private static void internalQuickSort(int[] a, int m, int n)  {
// quickSort()의 보조 메소드
    int p;
    if (m > n) then return;
    p = partition(a, m, n);  //한 피봇이 제 위치를 찾음
    internalQuickSort(a, m, p-1);
    internalQuickSort(a, p+1, n);
}

private static int partition(int[] a, int m, int n)  {
// internalQuicksort()의 보조 메소드
    • • • • •    // partition 알고리즘의 Java 코드
    return p;
}
```

퀵 정렬 (4)

- SortMain 클래스 (샘플 프로그램)

```
public class SortMain  {
    public static void main(String[] args)  {
        int[] a = { 5, 2, 8, 3, 1};  //이부분을 0에서 십만의 입력데이터생성
        System.out.println("정렬전 배열 원소 : ");
        int i;
        for (i = 0; i < a.length; i++)
            System.out.print(a[i] + "  ");
        System.out.println();
        Sorting. quickSort(a);     // 원하는 정렬 메소드를 선택
       // Sorting a1 = new Sorting();   //   ? 사용하지 마라
        //a1.quickSort(a);    //? 사용하지 말라.
        System.out.println("정렬된 배열 원소 : ");
        for (i = 0; i < a.length; i++)
            System.out.print(a[i] + " ");
        System.out.println();
    }
}
```

<실행결과>
정렬전 배열 원소 :
5 2 8 3 1
정렬된 배열 원소 :
1 2 3 5 8

합병 정렬 (4)

- mergeSort 프로그램(1)

```
public static void mergeSort(int[] a)  {
        // 합병 정렬의 메인 메소드
   int[] temp[a.length];  //원소의 개수에 비례하는 추가적인 공간
   internalMergeSort(a, temp, 0, a.length-1);
}

private static void internalMergeSort(int[] a, int[] temp, int m, int n)  {
        // 순환 호출을 하는 mergeSort()의 보조 메소드
   if (m < n)  {    // 정렬할 원소가 2개 이상인 경우
      int middle = (m+n) / 2;
      internalMergeSort(a, temp, m, middle);
      internalMergeSort(a, temp, middle+1, n);
      merge(a, temp, m, middle, middle+1, n);
   }
}
```

합병 정렬 (5)

- mergeSort 프로그램(2)

```
private static void merge(int[] a, int[] temp, int m, int p, int q, int n)  {
        // internalMergeSort()의 보조 메소드
   int t = m;
   int numElements = n – m + 1;
   while (m <= p && q <= n)  {
        if (a[m] < a[q])
            temp[t++] = a[m++];
        else temp[t++] = a[q++];
   }
   while (m <= p)     // 왼쪽 부분 배열에 원소가 남아 있는 경우
        temp[t++] = a[m++];
   while (q <= n)     // 오른쪽 부분 배열에 원소가 남아 있는 경우
        temp[t++] = a[q++];
   for (int i = 0,  i < numElements; i++, n--)     // 배열 temp[]를 a[]로 복사
       a[n] = temp[n];
}
```

- 단점 : 주어진 배열과 동일한 크기의 임시배열 **temp[]**가 필요

히프 정렬 (2)

- HeapSort 알고리즘

```
heapSort(a[])
   n ← a.length-1;   // n은 히프 크기(원소의 수)
                     // a[0]은 사용하지 않고 a[1 : n]의 원소를 오름차순으로 정렬
   for (i ← n/2;  i ≥ 1;  i ← i-1) do  {     // 배열 a를 히프로 변환
        heapify(a, i, n);                    // i는 내부 노드
   for (i ← n-1;  i ≥ 1;  i ← i-1) do  {     // 배열 a[]를 오름차순으로 정렬
        temp ← a[1];     // a[1]은 제일 큰 원소
        a[1] ← a[i+1];     // a[1]과 a[i+1]을 교환
        a[i+1] ← temp;
        heapify(a, 1, i);
   }
end HeapSort()
```

- Heapify()를 호출하여 배열 a[1 : n]을 히프 구조로 변환
- 원소를 교환하여 최대 원소 저장
- Heapify()를 호출하여 나머지 원소를 히프로 재구성

히프 정렬 (3)

- Heapify 알고리즘

```
heapify(a[], h, m)
        // 루트 h를 제외한 h의 왼쪽 서브트리와 오른쪽 서브트리는 히프
        // 현재 시점으로 노드의 최대 레벨 순서 번호는 m
  for (j ← 2*h;  j ≤ m;  j ← 2*j) do  {
    if (j < m) then
      if (a[j] < a[j+1]) then j ← j+1;    // j는 값이 큰 왼쪽 또는 오른쪽 자식 노드
    if (a[h] ≥ a[j]) then exit
    else a[j/2] ← a[j];     // a[j]를 부모 노드로 이동
  }
  a[j/2] ← a[h];
end heapify()
```

- 완전 2진 트리를 히프로 변환

Random Number

- Math.random
  a random double from 0.0 (inclusive) to 1.0(exclusive, 즉 0~0.999999)

```
1  public class MathRandom {
2
3      public static void main(String[] args) {
4          for (int i=0; i<10; ++i)
5              System.out.println(Math.random());
6      }
7  }
```

Random Number

- Java.util.Random.nextInt (랜덤 정수)

```
1  import java.util.Random;
2
3  public class RandomNextInt {
4      public static void main(String[] args) {
5          Random R = new Random();
6          for (int i=0; i<10; ++i)
7              System.out.println(R.nextInt(10));
8      }
9  }
```

Random Number

- Java.util.Random.ints (랜덤 정수)

  Random R1=new Random();

  R1.ints(min, max).findFirst().getAsInt();

  min(포함),max(제외) :

```
int nextInt(int origin, int bound) {
        int n = bound - origin;
        if (n > 0) { return nextInt(n) + origin; }
        else { // range not representable as
              int int r;
             do { r = nextInt();
             } while (r < origin || r >= bound);
            return r;
       }
```

Measuring Elapsed Time

```
1  import java.util.*;
2
3  public class ElapsedTime {
4
5      public static void main(String[] args) {
6          try {
7              long start = System.currentTimeMillis();
8              System.out.println(new Date());
9              Thread.sleep(5*1000);
10             System.out.println(new Date());
11             long end = System.currentTimeMillis();
12             long diff = end - start;
13             System.out.println("Difference: " + diff);
14         } catch (Exception e) {
15             System.out.println("An exception occurs");
16         }
17     }
18 }
```

1. (퀵, 히프,합병)정렬을 완성하라.
2. 실행속도와 관련된 엑셀 그래프를 완성하라.

Upload your files at smartlead.hallym.ac.kr

감사합니다.