Sorting

โครงสร้างข้อมูลและอัลกอริทึมเบื้องต้น
305214 / 235012

Topic

- Selection Sort
- Bubble Sort
- Merge Sort
- Quick Sort
- Heap Sort

Sorting

การเรียงข้อมูล

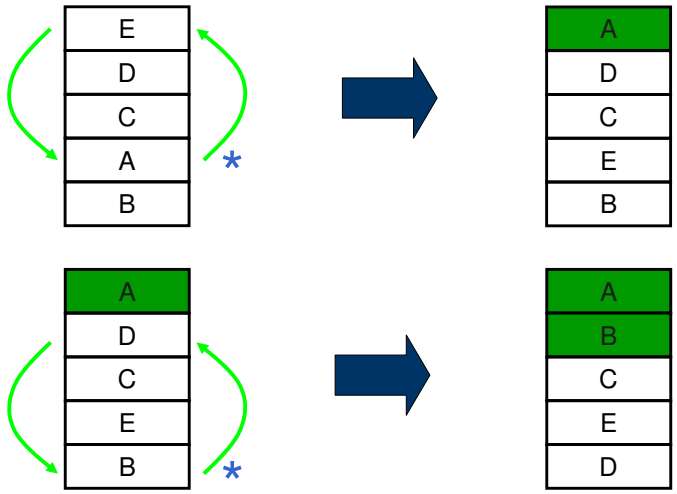
- ช่วยให้การค้นหาทำได้เร็วขึ้น
- เป็นการสลับตำแหน่งข้อมูลที่เก็บไว้จากน้อยไปมาก (Ascending) หรือมากไปน้อย (Descending)
  - การเรียงแบบเลือก Selection Sort
  - การเรียงแบบฟอง Bubble Sort
  - การเรียงแบบผสาน Merge Sort
  - การเรียงแบบเร็ว Quick Sort
  - การเรียงแบบฮีป Heap Sort

Selection Sort

การเรียงแบบเลือก

- ทำการเปรียบเทียบข้อมูลโดยเริ่มจากตัวแรกเปรียบเทียบกับตัวที่เหลือทีละตัวเพื่อหาค่าน้อยที่สุด
- ได้ค่าน้อยที่สุดก่อน ทำการสลับกับข้อมูลตำแหน่งแรก
- ทำกับข้อมูลชุดที่เหลือเหมือนเดิม
- วนทำต่อไปจนหมดตามจำนวนข้อมูล

Selection Sort
การเรียงแบบเลือก



Selection Sort ตัวอย่าง

```
void selection_sort(char c_array[], int count)
{       int i, pass, index;
        char temp;
        for(pass=0;pass<count-1;pass++)
        {
                index=pass;
                for(i=pass+1;i<count;i++)
                        if(c_array[i]<c_array[index]
                                index=i;
                if(index!=pass)
                {
                        temp=c_array[pass];
                        c_array[pass]=c_array[index];
                        c_array[index]=temp;
                }                                       }}
```


Selection Sort
Algorithm Analysis

มีข้อมูล N ตัว

รอบที่ 1 จำนวนครั้งการเปรียบเทียบ → N - 1

รอบที่ 2 จำนวนครั้งการเปรียบเทียบ → N - 2

จำนวนครั้งทั้งหมด → (N - 1) + (N – 2) + … + 1

- ได้ค่าเป็น $\frac{(N-1)N}{2}$
- Big O $O(N^2)$

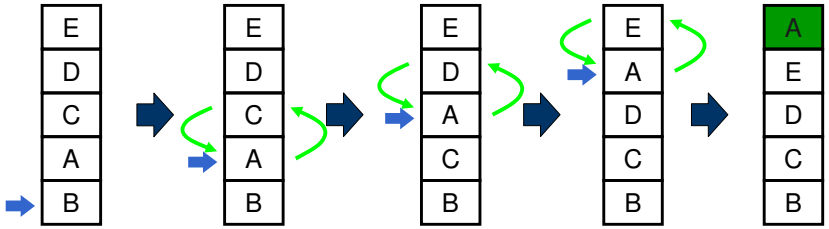
Bubble Sort

การเรียงแบบฟอง

- ทำการเปรียบเทียบข้อมูลโดยเริ่มจากตัวท้ายเปรียบเทียบกับตัวก่อนหน้า หากน้อยกว่าทำการสลับ
- ทำการเลื่อนทำตำแหน่งก่อนหน้า ไปเรื่อยๆ จนถึงตำแหน่งแรก ได้ตำแหน่งแรกเป็นค่าน้อยที่สุด
- วนรอบทำที่ตำแหน่งสุดท้ายใหม่กับข้อมูลชุดที่เหลือ
- วนทำต่อไปจนหมดตามจำนวนข้อมูล

Bubble Sort

การเรียงแบบฟอง



เหมือนฟองอากาศในน้ำ ค่าที่น้อยที่สุดลอยขึ้นข้างบน

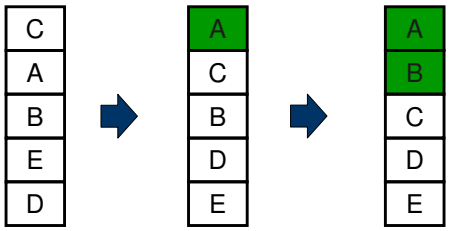
Bubble Sort ตัวอย่าง

```
void bubble_sort(char c_array[], int count)
{       int i, index;
        char temp;
        for(i=1;i<count;i++)
                for(index=count-1;index>=i;index--)
                        if(c_array[index]<c_array[index-1])
                        {
                                temp=c_array[index];
                                c_array[index]=c_array[index-1];
                                c_array[index-1]=temp;
                        }
}
```

swop

Bubble Sort

ตัวอย่างการเรียงแบบฟองกรณีข้อมูลเกือบเรียง



ควรมีการตรวจสอบ หากไม่มีการสลับอีกเลยให้เลิกทำ

Bubble Sort ตัวอย่าง 2

```
void bubble_sort(char c_array[], int count)
{       int i, index, finish;
        char temp;    finish=0;   i=1;
        while((i<count)&&(finish==0))
        {
                finish=1;
                for(index=count-1;index>=i;index--)
                        if(c_array[index]<c_array[index-1])
                        {
                                temp=c_array[index];
                                c_array[index]=c_array[index-1];
                                c_array[index-1]=temp;
                                finish=0;
                        }
                i++;                    }}
```

Exit while loop

Bubble Sort

Algorithm Analysis

เหมือนกับ Selection Sort

มีข้อมูล N ตัว

จำนวนครั้งทั้งหมด → (N - 1) + (N – 2) + … + 1

- ได้ค่าเป็น $\frac{(N-1)N}{2}$
- Big O $O(N^2)$
- ยกเว้นกรณีที่มีการใช้ flag : best case เป็น O(N)

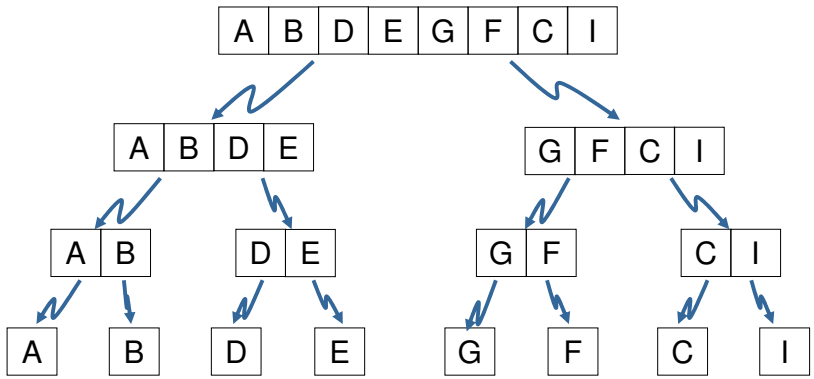
Merge Sort

การเรียงแบบผสาน

- ทำการแบ่งข้อมูลเป็นสองส่วน
- แต่ละส่วนนำไปจัดเรียงกัน
- นำมาผสานกันโดยไล่ไปยังข้อมูลแต่ละส่วน โดยเลือกค่าน้อยมาก่อน
- ใช้วิธีการ recursive โดยทำการแบ่งข้อมูลจนเหลือเพียงข้อมูลเดียวแล้วทำการผสานกันกลับขึ้นไป

Merge Sort

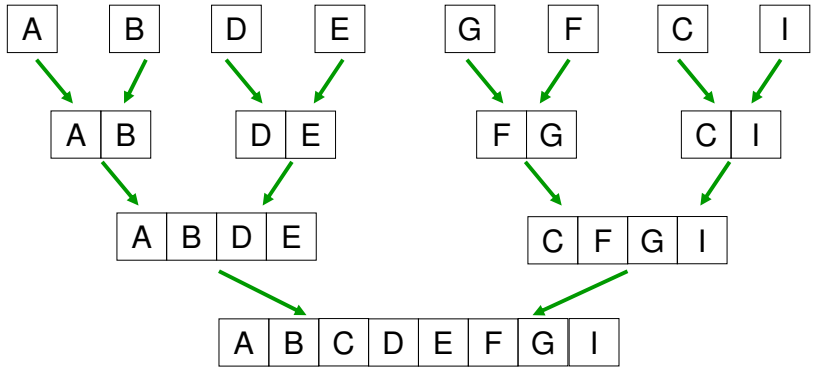
การเรียงแบบผสาน



แบ่งครึ่งจนเหลือข้อมูลย่อยที่สุด

Merge Sort

การเรียงแบบผสาน (ต่อ)



ทำการผสาน (merge) กลับโดยมีการจัดเรียง

Merge Sort ตัวอย่าง merge sort

```
void merge_sort(array_list *c_arr,int first, int last)
{
        int mid;
        if(first<last)
        {
                mid=(first+last)/2;
                merge_sort(c_arr,first,mid);
                merge_sort(c_arr,mid+1,last);
                merge(c_arr,first,mid,mid+1,last);
        }
}
```


Merge Sort ตัวอย่าง merge (1)

```
void merge(char c_arr,int left_first,int left_last,int right_first,
int right_last)
{
        char temp[max_c_arr_size];
        int index, left_first_idx;
        index=left_first;
        left_first_idx=left_first;
```


Merge Sort ตัวอย่าง Merge (2)

```
        while((left_first<=left_last)&&(right_first<=right_last))
        {
                if(c_arr[left_first]<c_arr[right_first])
                {
                        temp[index]=c_arr[left_first];
                        left_first++;
                } else
                {
                        temp[index]=c_arr[right_first];
                        right_first++;
                }
                index++;
        }
```


Merge Sort ตัวอย่าง Merge (3)

```
        while(left_first<=left_last)
        {       temp[index]=c_arr[left_first];
                left_first++;
                index++;
        }
        while(right_first<=right_last)
        {       temp[index]=c_arr[right_first];
                right_first++;
                index++;
        }
        for(index=left_first_idx;index<=right_last;index++)
                c_arr[index]=temp[index];
}
```

Merge Sort

Algorithm Analysis

นับจำนวนครั้งในการเปรียบเทียบแต่ละชั้น

- ชั้นแรก เปรียบเทียบ N – 1 ครั้ง จำนวน list 1
- ชั้นสอง เปรียบเทียบ N/2 – 1 ครั้ง จำนวน list 2
- ชั้นสาม เปรียบเทียบ N/4 – 1 ครั้ง จำนวน list 4

. . .

- ชั้นก่อนท้าย เปรียบเทียบ 2 – 1 ครั้ง จำนวน list N/2
- ชั้นสุดท้าย เปรียบเทียบ 0 ครั้ง จำนวน list N

Merge Sort

Algorithm Analysis (2)

รวมจำนวนครั้งในการเปรียบเทียบ

$= (N - 1)*1 + (N/2 - 1)*2 + (N/4 - 1)*4 + \ldots + (2-1)*N/2$

$= N - 1 + N - 2 + N - 4 + \ldots + N - N/2$

*จำนวนชั้นเป็น $\log_2 N$ ดังนั้นเมื่อมี N อยู่เท่ากับจำนวนชั้น

$= N - 1 + N - 2 + N - 4 + \ldots + N - N/2$

$\frac{a_1 - a_n r}{1 - r}$

$= N \log_2 N - (1+2+4+\ldots+N/2)$

$= N \log_2 N - (N - 1)$ → **Big O** = $O(N \log_2 N)$

Big-O Notation

การเพิ่มค่า N

N	$\text{Log}_2 N$	$N \text{Log}_2 N$	N^2	N^3	2^N
1	0	0	1	1	2
2	1	2	4	8	4
4	2	8	16	64	16
8	3	24	64	512	256
16	4	64	256	496	65536
32	5	160	1024	32768	2147483648
64	6	384	4096	262144	1.8447E+19
128	7	896	16384	2097152	3.4028E+38
256	8	2048	65536	16777216	1.1579E+77

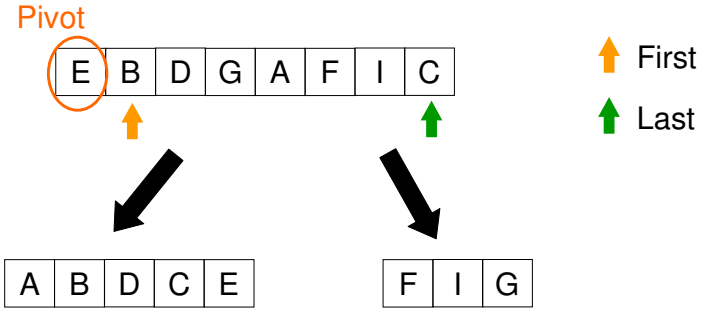
Quick Sort

การเรียงแบบเร็ว

- แบ่งข้อมูลเป็นสองส่วนโดยข้อมูลใดข้อมูลหนึ่ง
- ทำการสลับเพื่อย้ายข้อมูลให้ข้อมูลที่เลือกไปอยู่ที่จุดแบ่งข้อมูลที่น้อยกว่าอยู่ทางซ้าย ข้อมูลที่มากกว่าอยู่ทางขวา
- ทำการเรียงข้อมูลที่แบ่งออกมาทั้งซ้ายและขวา
- วนกระทำจนแบ่งข้อมูลอีกต่อไปไม่ได้

Quick Sort

การเรียงแบบเร็ว



แบ่งส่วนนำไปทำกระบวนการต่อ

Quick Sort ตัวอย่าง

```
void quick_sort(char c_arr[], int first, int last)
{
        int splitpoint;
        if(first<last)
        {
                split(c_arr,first,last,&splitpoint);
                quick_sort(c_arr,first,splitpoint-1);
                quick_sort(c_arr,splitpoint,last);
        }
}
```


Quick Sort ตัวอย่าง split (1)

```
void split(char c_arr[], int first, int last, int *splitpoint)
{
        int pivot, first_idx, less_pivot, more_pivot;

        pivot=c_arr[first];
        first_idx=first;
        first++;
        do
        {       less_pivot=1;
                while((first<=last)&&(less_pivot==1))
                        if(c_arr[first] > pivot)
                                less_pivot=0;
                        else
                                first++;
```


Quick Sort ตัวอย่าง split (2)

```
                more_pivot=1;
                while((first<=last)&&(more_pivot==1))
                        if(pivot>c_arr[last])
                                more_pivot=0;
                        else
                                last--;
                if(first<last)
                {       swap(&c_arr[first],&c_arr[last]);
                        first++;
                        last--;
                }
        }while(first<=last);
        *splitpoint=last;

        swap(&c_arr[first_idx],&c_arr[*splitpoint]);    }
```

Quick Sort

Algorithm Analysis

จำนวนครั้งในการเปรียบเทียบขึ้นอยู่กับลักษณะการจัดเรียงและการเลือก pivot

- เลือก pivot ได้ค่ากลาง จะได้กรณีดีที่สุดคือแบ่งครึ่งพอดี

  ได้ $O(N \log_2 N)$ คล้ายกับ Merge Sort

- เลือก pivot ได้ค่าน้อยหรือมากที่สุดและข้อมูลเรียงกันอยู่แล้ว

  เป็นกรณีแย่ที่สุดมีการแบ่งทุกครั้ง ได้เป็น $O(N^2)$

  คล้ายกับ select sort

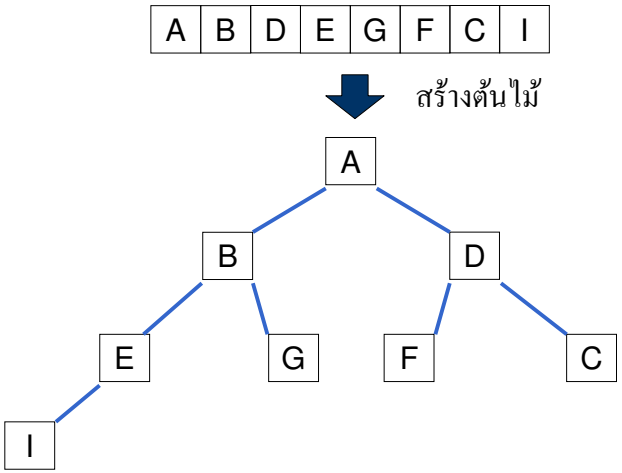
Heap Sort

การเรียงแบบ heap

- สร้างข้อมูล list ที่ต้องการเรียงให้เป็นต้นไม้
- สร้างต้นไม้ฮีป
- สลับ root กับ ท้ายสุด
- สร้าง heap ใหม่ยกเว้น root ที่สลับไป
- วนทำเรื่อยๆ จนจบ (หมด node)

Heap Sort

การสร้าง binary tree จาก list



Heap Sort

การสร้าง heap tree

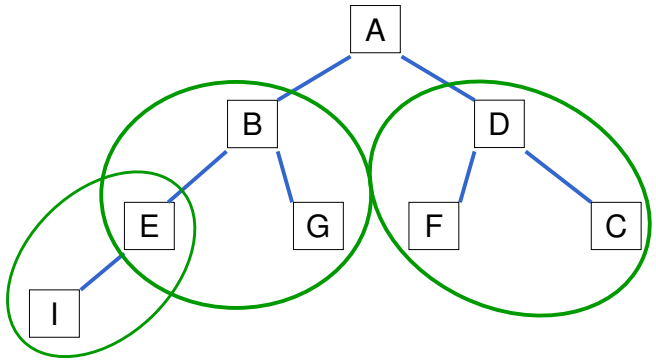
คุณสมบัติของ heap tree

- เป็นต้นไม้ทวิภาค (binary tree) เต็มต้น
- โหนด Parent ต้องมีค่ามากว่า โหนด Child

ต้นไม้ที่ได้จาก list เต็มต้นอยู่แล้ว เพียงแต่ต้องนำมาเรียงโหนดใหม่ให้ parent มากกว่า child

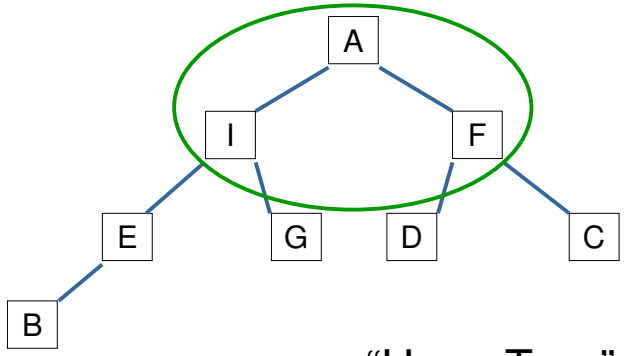
Heap Sort

การสร้าง heap tree (2)



Heap Sort

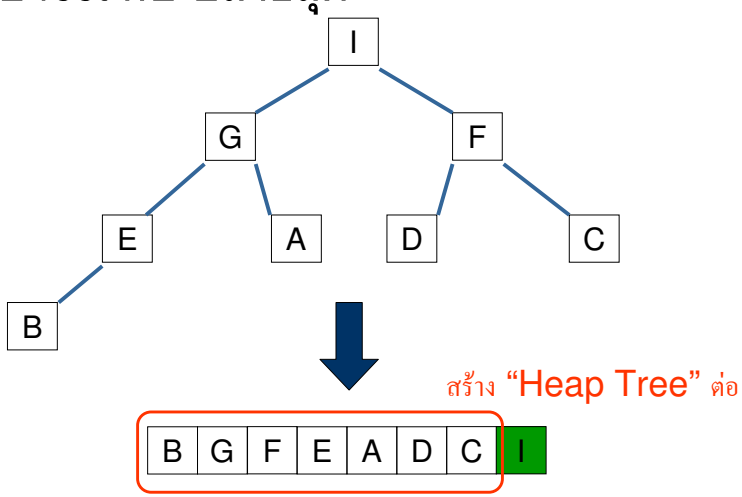
การสร้าง heap tree (3)



"Heap Tree"

Heap Sort

สลับ root กับ ปลายสุด



Heap Sort ตัวอย่าง

```
void heap_sort(char c_arr[], int count)
{
        int i;
        for(i=(count/2)-1;i>=0;i--)
                reheap(c_arr, i,count);     ทำให้เป็น Heap
        for(i=count-1;i>=1;i--)
        {
                swap(&c_arr[0],&c_arr[i]);
                reheap(c_arr,0,i);
        }
}
```

Heap Sort ตัวอย่าง reheap

```
void reheap(char c_arr[], int root, int count)
{
        int maxchild, IsHeap;
        IsHeap=0;
        while( ((root+1) * 2 <= count) && (IsHeap==0) )
        {
                if((root+1)*2 == count)
                        maxchild=(root+1)*2-1;
                else
                        if(c_arr[(root+1)*2]>c_arr[(root+1)*2+1])
                                maxchild=(root+1)*2-1;
                        else
                                maxchild=(root+1)*2;
```

เทียบหา Child ที่มากกว่า

Heap Sort ตัวอย่าง reheap (ต่อ)

```
                if(c_arr[root]<c_arr[maxchild])
                {
                        swap(&c_arr[root],&c_arr[maxchild]);
                        root=maxchild;
                } else
                        IsHeap=1;
        }
}
```

เทียบหา Child ที่มากกว่า กับ root และทำการสลับ

Heap Sort

Algorithm Analysis

จำนวนครั้งในการทำงานนับจากจำนวนครั้งในการเปรียบเทียบคูณกับรอบการทำงานของการ reheap

- ครั้งในการเปรียบเทียบเท่ากับ O(N-1) = O(N)
- จำนวนรอบการทำงานของ reheap เท่ากับความสูง tree

  เป็น $\log_2 N = O(\log_2 N)$

  ดังนั้น Heap Sort เป็น $O(N \log_2 N)$

Summary

Algorithm Analysis

Sorting	Best	Average	Worst
Select	$O(N^2)$	$O(N^2)$	$O(N^2)$
Bubble	$O(N^2)$	$O(N^2)$	$O(N^2)$
Bubble + flag	$O(N)$	$O(N^2)$	$O(N^2)$
Merge	$O(N \log_2 N)$	$O(N \log_2 N)$	$O(N \log_2 N)$
Quick	$O(N \log_2 N)$	$O(N \log_2 N)$	$O(N^2)$
Heap	$O(N \log_2 N)$	$O(N \log_2 N)$	$O(N \log_2 N)$