2110211 โครงสร้างข้อมูลเบื้องต้น

Sorting Algorithms

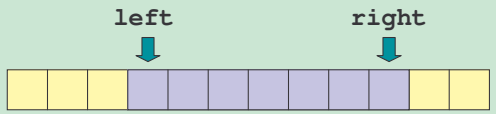
สมชาย ประสิทธิ์จูตระกูล

Comparison-Based Sorting Algorithms

- Bubble sort $O(n^2)$
- Selection sort $O(n^2)$
- Insertion sort $O(n^2)$
- Shell sort $O(n^{1.xx})$
- Merge sort $O(n \log n)$
- Heap sort $O(n \log n)$
- Quick sort $O(n \log n)$ avg.

อาศัยการนำข้อมูลมาเปรียบเทียบกันทีละคู่ (Radix sort ไม่ใช่)

```
static void xxSort(Object[] d, int left, int right) {
  ...
  if ( ((Comparable)d[i]).compareTo(d[j]) < 0 ) {
    ...
  } else {
    ...
  }
  ...
```



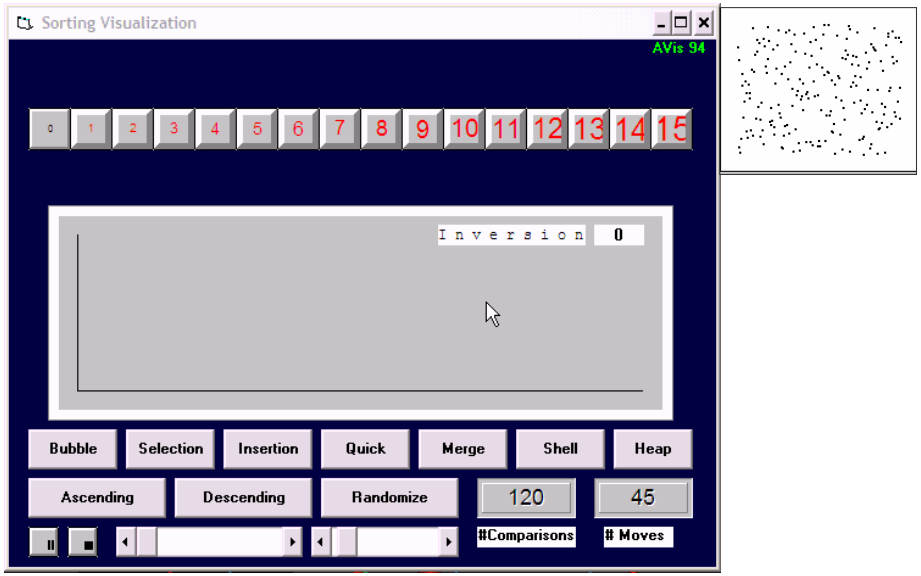
การกลับลำดับ (Inversion)

- d[i] และ d[j] กลับลำดับ เมื่อ i<j แต่ d[i]>d[j]
- เรียงจากน้อยไปมาก จำนวนการกลับลำดับ=0
- เรียงจากมากไปน้อย จำนวนการกลับลำดับ=n(n-1)/2

lessThan, swap

```
public class ArrayUtil {

  private static boolean lessThan(Object a, Object b) {
    return ((Comparable)a).compareTo(b) < 0;
  }

  private static void swap(Object[] d, int i, int j) {
    Object t = d[i]; d[i] = d[j]; d[j] = t;
  }
  ...
}
```

Selection Sort



Selection Sort

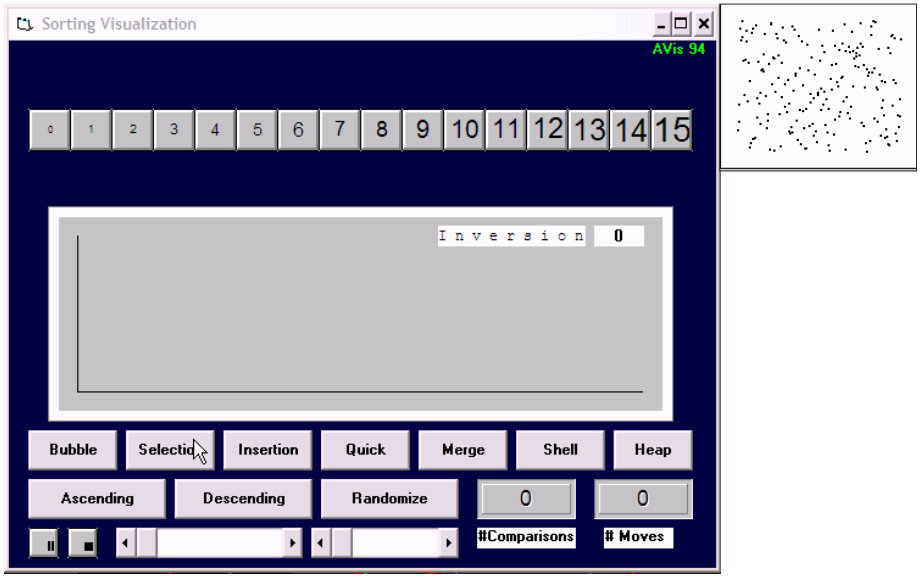
```
public class ArrayUtil {

  public static void
  selectionSort(Object[] d, int left, int right) {

    for(int i = right; i > left; i--) {
      int m = i;
      for(int j = left; j < i; j++) {
        if (lessThan(d[m],d[j])) m = j;
      }
      swap(d, m, i);  // d[m] ←→ d[k]
    }

  }
  ...
}
```

รอบที่	จำนวนข้อมูล	#(lessThan)
1	n	n-1
2	n-1	n-2
...	...	...
n-1	2	1

$\sum = n(n-1)/2$

#swaps = n-1

Bubble Sort (exchange sort)



Bubble Sort

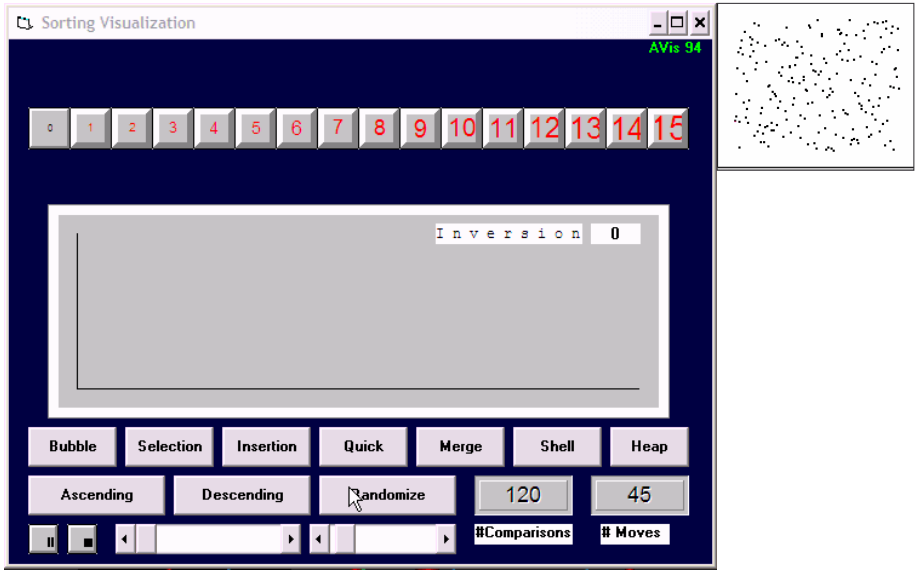
```
public class ArrayUtil {

  public static void
  bubbleSort(Object[] d, int left, int right) {

    for (int i = right; i > left; i--) {
      for (int j = left; j < i; j++) {
        if (lessThan(d[j+1],d[j])) swap(d, j+1, j);
      }
    }

  }
  ...
}
```

รอบที่	จำนวนข้อมูล	#(lessThan)
1	n	n-1
2	n-1	n-2
...	...	...
n-1	2	1

$\sum = n(n-1)/2$

$0 \le \#swaps \le n(n-1)/2$

Insertion Sort



Insertion Sort

```
public class ArrayUtil {

  public static void
  insertionSort(Object[] d, int left, int right) {

    for (int i = left+1; i <= right; i++) {
      Object t = d[i];
      int j = i-1;
      while( j >= left && lessThan(t, d[j]) ) {
         d[j+1] = d[j];
         j--;
      }
      d[j+1] = t;
    }

  }
  ...
}
```

ยิ่งเรียงยิ่งเร็ว

รอบที่	#(lessThan)	
1	≤ 1	∑ ≤ n(n-1)/2
2	≤ 2	
...	...	
n-1	≤ n-1	

$2(n-1) \le \#moves \le (n+4)(n-1)/2$

Shell Sort (Donald Shell, 1959)

- Insertion Sort
  - ช้าเพราะเลื่อนข้อมูลไปช่องถัดไป (ทีละตำแหน่ง)
  - ลด inversion ทีละหนึ่ง ต่อการเลื่อนหนึ่งครั้ง
- Shell Sort
  1. แบ่งข้อมูลเป็น h ชุด แบบตัวเว้น h – 1 ตัว
  2. sort แต่ละชุดด้วย insertion sort
  3. ถ้า h == 1 ก็เสร็จ ไม่เช่นนั้น ลดค่า h ลง กลับไปข้อ 1
- หมายเหตุ
  - รอบสุดท้ายเป็น insertion sort ชุดเดียว
  - ยังไม่มีใครรู้ว่าลด h อย่างไรถึงดีสุด !

Shell Sort : ปรับจาก insertion sort

sort แต่ละชุด

insertion sort with h increment

```
shellSort(Object[] d, int left, int right) {
  for (int h = ???; h > 0; h = ???) {
    for (int m = 0; m < h; m++) {
      for (int i = left+m+h; i <= right; i+=h) {
        Object t = d[i];  int j = i-h;
        while( j >= left && lessThan(t, d[j]) ) {
          d[j+h] = d[j];  j -= h;
        }
        d[j+h] = t;
      }
    }
  }
}
```

```
insertionSort(Object[] d, int left, int right) {
  for (int i = left+1; i <= right; i++) {
    Object t = d[i];  int j = i-1;
    while( j >= left && lessThan(t, d[j]) ) {
       d[j+1] = d[j];  j--;
    }
    d[j+1] = t;
  }
}
```

Shell Sort : การเขียนโปรแกรม

```
shellSort(Object[] d, int left, int right) {
  for (int h = ???; h > 0; h = ???) {
    for (int m = 0; m < h; m++) {
      for (int i = left+m+h; i <= right; i+=h) {
        ...
      }
    }
  }
}
```

sort ให้เสร็จทีละชุด

```
shellSort(Object[] d, int left, int right) {
  for (int h = ???; h > 0; h = ???) {

    for (int i = left+h; i <= right; i++) {
      ...
    }

  }
}
```

สลับ sort ทั้ง k ชุดไปพร้อม ๆ กัน

Shell Sort : ดีจริงหรือ ?

0	1	2	3	4	5	6	7	8	9	10	11
11	6	9	4	10	8	2	5	7	3	0	1
0	1	2	3	4	5	6	7	8	9	10	11

insertion sort ปกติ

59 cmps

11	6	9	4	10	8	2	5	7	3	0	1

แบ่งเป็น 2 ชุด

11	6	9	4	10	8	2	5	7	3	0	1

sort ชุดคู่ 15 cmps

sort ชุดคี่ 14 cmps

0	1	2	3	7	4	9	5	10	6	11	8

0	1	2	3	7	4	9	5	10	6	11	8

sort ชุดเดียวอีก 20 cmps

0	1	2	3	4	5	6	7	8	9	10	11

รวม 49 cmps

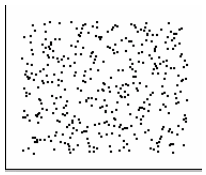
Shell Sort : h-sequence อะไรดี ?

- ยังไม่มีผู้ใดพบ h-sequence ที่ดีสุด
- ชุดที่ดีมักมีค่าลดลงเป็นเท่า ๆ
  - Shell : 1,2,4,8, ..., 2^m,... $O(n^2)$
  - Hibbard : 1,3,7,15, ..., $2^m - 1$,... $O(n^{3/2})$
  - Knuth : 1,4,13,40,121, ..., $(3^m - 1)/2$,... $O(n^{3/2})$
  - Sedgewick : 1,8,23,77, ..., $4^{m+1}+3\cdot 2^m+1$,... $O(n^{4/3})$

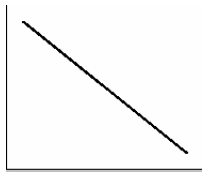
```
shellSort(Object[] d, int left, int right) {
  int k;
  for (k = 1; k <= (right-left)/9; k = 3*k+1);
  for ( ;k > 0; k /= 3) {
    for (int i = left+k; i <= right; i++) {
      ...
    }
  }
}
```

Knuth's sequence

Shell Sort : Animations



random

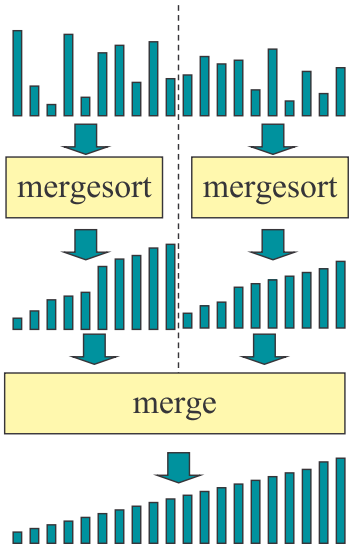


inverse

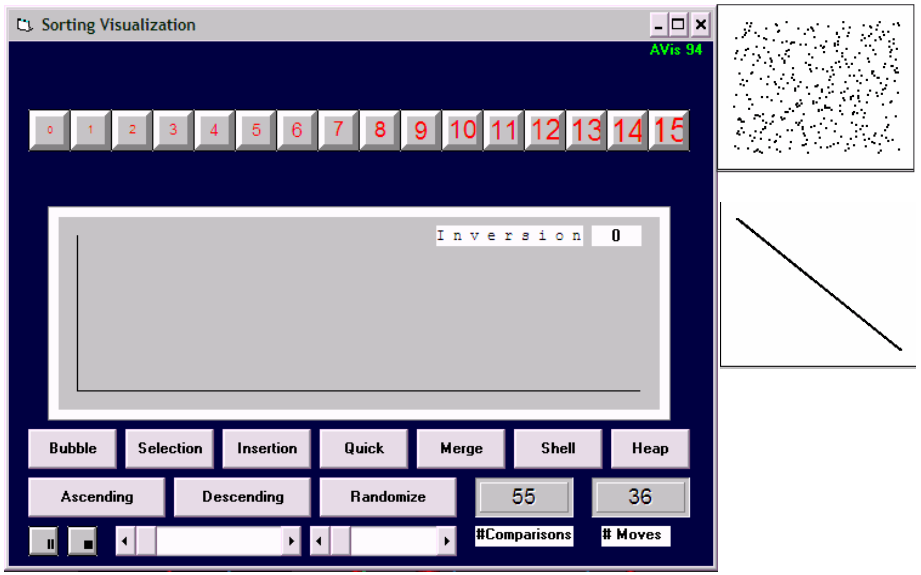
การบ้าน

- ปรับปรุง bubble sort ให้เร็วขึ้น โดยตรวจสอบว่าการวิ่ง loop ในถ้าไม่เกิดการ swap เลย ก็ย่อมแสดงเรียงเรียบร้อยแล้ว
- ยกตัวอย่างชุดข้อมูลที่ทำให้ Shell sort ที่ใช้ h-sequence ของ Shell (1,2,4,8, ..., 2^m,...) แล้วการทำงานแย่สุด ๆ
- หลักการทำงานของ selection sort เหมือนกับ heap sort อย่างไร

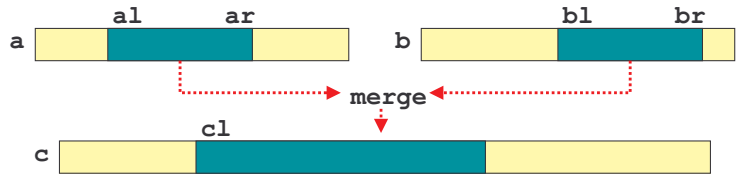
Merge Sort : Divide & Conquer



Merge Sort



Merge ให้เป็นก่อน



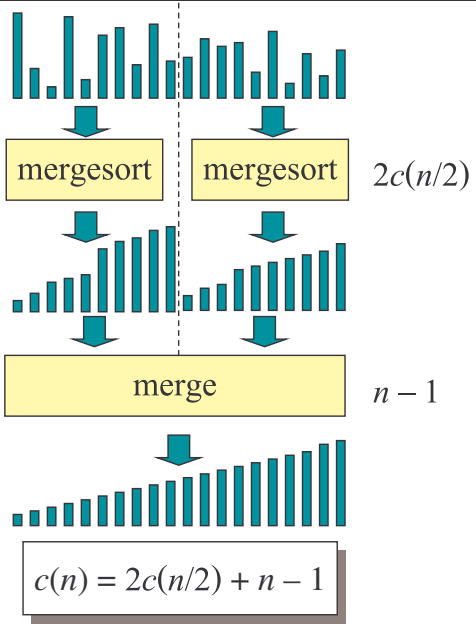

```
private static void merge(Object[] a, int al, int ar,
                          Object[] b, int bl, int br,
                          Object[] c, int cl ) {
  int i = al, j = bl, n = (ar-al+1)+(br-bl+1);
  for (int k = cl; k < cl+n; k++) {
    if (i > ar) { c[k] = b[j++]; continue; }
    if (j > br) { c[k] = a[i++]; continue; }
    c[k] = lessThan(a[i], b[j]) ? a[i++] : b[j++];
  }
}
```

a และ b เป็นอาเรย์ตัวเดียวกันก็ได้ ถ้า [al,ar] ไม่ซ้อนทับกับ [bl,br]

Merge Sort : Recursive

```
public class ArrayUtil {

  public static void
  mergeSort(Object[] d, int left, int right) {
    mergeSortR(d, left, right, d.clone());
  }
  public static void
  mergeSortR(Object[] d, int left, int right, Object[] t) {
    if (left < right) {
      int m = (left + right) / 2;
      mergeSortR(t, left, m, d);
      mergeSortR(t, m + 1, right, d);
      merge(t, left, m, t, m + 1, right, d, left);
    }
  }

  ...
}
```

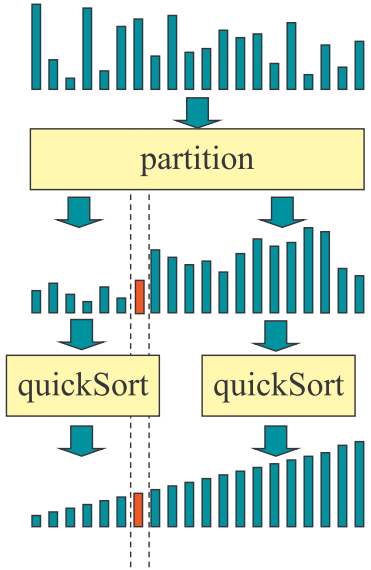

Merge Sort : Analysis

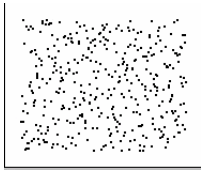


Merge Sort : Analysis

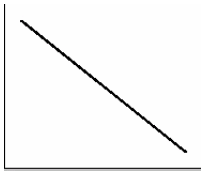
$$
\begin{aligned}
c(n) &= 2c(n/2) + (n-1) \quad n > 1,\ c(0) = c(1) = 0, \quad n = 2^k \\
&= 2\left(2c\left(n/2^2\right) + (n/2 - 1)\right) + (n-1) \\
&= 2^2 c\left(n/2^2\right) + (n-2) + (n-1) \\
&= 2^3 c\left(n/2^3\right) + (n-4) + (n-2) + (n-1) \\
&\cdots \\
&= 2^k c\left(n/2^k\right) + kn - \left(2^{k-1} + 2^{k-2} + \cdots + 2^0\right) \\
&= n\log n - n + 1 \\
&= O(n\log n)
\end{aligned}
$$

Quick Sort (C. A. R. Hoare, 1961)





random

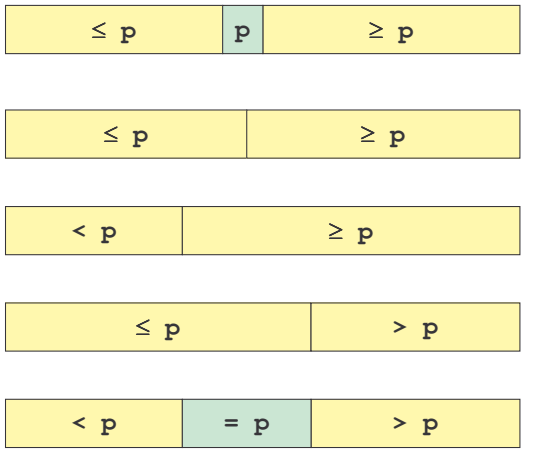


inverse

Quick Sort : Recursive

```
public class ArrayUtil {

  public static void
  quickSort(Object[] d, int left, int right) {
    if (left < right) {
      int i = partition(d, left, right);
      quicksort(d, left, i-1);
      quicksort(d, i+1, right);
    }
  }

  ...
}
```

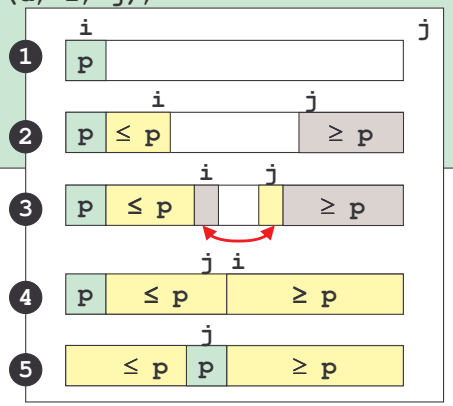

Quicksort : Partitioning



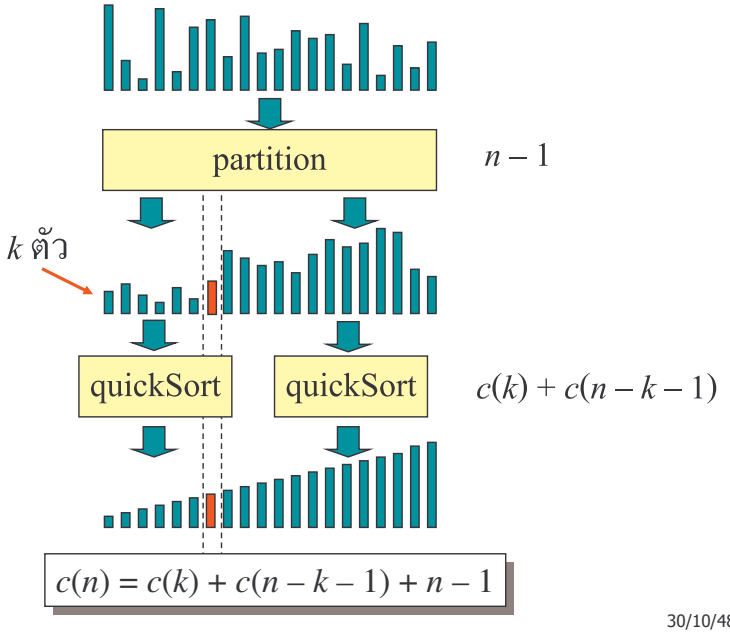
p - pivot

Quick Sort : Partitioning

```
private static
int partition(Object[] d, int left, int right) {
  Object p = d[left];
1 int i = left, j = right + 1;
2 while (i < j) {
    while (lessThan(p, d[--j])) ;
    while (lessThan(d[++i], p)) if (i == right) break;
3   if (i < j) swap(d, i, j);
  }
4
  swap(d,left,j);
5 return j;
}
```



Quick Sort : Analysis



Quick Sort : Analysis

- c(n) = c(k) + c(n – k – 1) + (n – 1)
- best case : k = n/2
  - c(n) = 2c(n/2) + (n – 1) = O(n log n)
- worst case : k = 1
  - c(n) = c(0)+c(n – 1)+(n–1) = c(n – 1)+(n–1) = O(n^2)
- ถ้าเราเลือก pivot แบบสุ่ม โอกาสจะเป็นตัวใดๆ = 1/n

$$c(n) = (n-1) + \frac{1}{n}\sum_{k=0}^{n-1}\big(c(k) + c(n-k-1)\big)$$

Randomized QuickSort : Analysis

$$c(n) = (n-1) + \frac{1}{n}\sum_{k=0}^{n-1}\big(c(k) + c(n-k-1)\big) = (n-1) + \frac{2}{n}\sum_{k=0}^{n-1} c(k)$$

$$nc(n) = n(n-1) + 2\sum_{k=0}^{n-1} c(k)$$

$$(n-1)c(n-1) = (n-1)(n-2) + 2\sum_{k=0}^{n-2} c(k)$$

$$nc(n) - (n-1)c(n-1) = 2(n-1) + 2c(n-1) + c(0)$$

$$nc(n) = (n+1)c(n-1) + 2(n-1)$$

$$\frac{c(n)}{n+1} = \frac{c(n-1)}{n} + \frac{2(n-1)}{n(n+1)} = \frac{c(n-2)}{n-1} + \frac{2(n-2)}{(n-1)n} + \frac{2(n-1)}{n(n+1)}$$

$$\cdots$$

$$= \frac{c(n-n)}{1} + \sum_{i=1}^{n}\frac{2(i-1)}{i(i+1)} = 2\ln n$$

$$c(n) = 2(n+1)\ln n \approx 1.38\log_2 n = O(n\log n)$$

Tuned Quicksort (ใช้ใน java.util.Arrays)

- ใช้ insertion sort เมื่อ n < 7
- ถ้า 7 ≤ n < 40
  - เลือก pivot จาก median ของตัวซ้าย กลาง ขวา
- ถ้า n > 40
  - เลือก pivot จาก median of median of three ของข้อมูล 9 ตัว
- partition แบบ 3 ช่วง <p =p >p

Jon L. Bentley and M. Douglas McIlroy's "Engineering a Sort Function", Software-Practice and Experience, Vol. 23(11) P. 1249-1265 (November 1993).

Sorting Animations