Advanced Topics in Sorting

anhtt-fit@mail.hut.edu.vn
dungct@it-hut.edu.vn

http://www.4shared.com/file/79096214/fb2ed224/lect01.html

Sorting applications

Sorting algorithms are essential in a broad variety of applications

- Organize an MP3 library.
- Display Google PageRank results.
- List RSS news items in reverse chronological order.
- Find the median.
- Find the closest pair.
- Binary search in a database.
- Identify statistical outliers.
- Find duplicates in a mailing list.
- Data compression.
- Computer graphics.
- Computational biology.
- Supply chain management.
- Load balancing on a parallel computer.
- . . .

Sorting algorithms

Many sorting algorithms to choose from

Internal sorts

- Insertion sort, selection sort, bubblesort, shaker sort.
- Quicksort, mergesort, heapsort, samplesort, shellsort.
- Solitaire sort, red-black sort, splaysort, Dobosiewicz sort, psort, ...

External sorts

- Poly-phase mergesort, cascade-merge, oscillating sort.

Radix sorts

- Distribution, MSD, LSD.
- 3-way radix quicksort.

Parallel sorts

- Bitonic sort, Batcher even-odd sort.
- Smooth sort, cube sort, column sort.
- GPUsort.

Which algorithm to use?

Applications have diverse attributes

- Stable?
- Multiple keys?
- Deterministic?
- Keys all distinct?
- Multiple key types?
- Linked list or arrays?
- Large or small records?
- Is your file randomly ordered?
- Need guaranteed performance?

Cannot cover all combinations of attributes.

Case study 1

Problem

- Sort a huge randomly-ordered file of small records.

Example

- Process transaction records for a phone company.

Which sorting method to use?

1. Quicksort: YES, it's designed for this problem
2. Insertion sort: No, quadratic time for randomly-ordered files
3. Selection sort: No, always takes quadratic time

Case study 2

Problem

- Sort a huge file that is already almost in order.

Example

- Re-sort a huge database after a few changes.

Which sorting method to use?

1. Quicksort: probably no, insertion simpler and faster
2. Insertion sort: YES, linear time for most definitions of "in order"
3. Selection sort: No, always takes quadratic time

Case study 3

Problem: sort a file of huge records with tiny keys.

Ex: reorganizing your MP3 files.

Which sorting method to use?

1. Mergesort: probably no, selection sort simpler and faster
2. Insertion sort: no, too many exchanges
3. Selection sort: YES, linear time under reasonable assumptions

Ex: 5,000 records, each 2 million bytes with 100-byte keys.

- Cost of comparisons: $100 \times 5000^2 / 2 = 1.25$ billion
- Cost of exchanges: 2,000,000 x 5,000 = 10 trillion
- Mergesort might be a factor of log (5000) slower.

Duplicate keys

Often, purpose of sort is to bring records with duplicate keys together.

- Sort population by age.
- Finding collinear points.
- Remove duplicates from mailing list.
- Sort job applicants by college attended.

Typical characteristics of such applications.

- Huge file.
- Small number of key values.

Mergesort with duplicate keys: always ~ N lg N compares

Quicksort with duplicate keys

- algorithm goes quadratic unless partitioning stops on equal keys!
- 1990s Unix user found this problem in qsort()

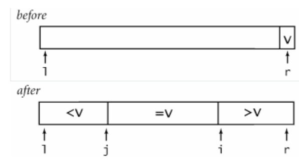
Exercise: Create Sample Data

- Write a program that generates more than 1 million integer numbers. These number are in range of 40 different discrete values.

3-Way Partitioning

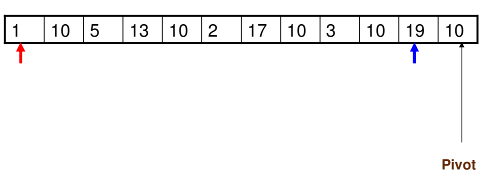
3-way partitioning. Partition elements into 3 parts:

- Elements between i and j equal to partition element v.
- No larger elements to left of i.
- No smaller elements to right of j.



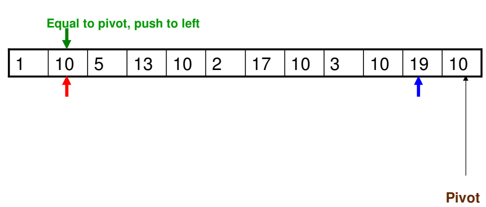
Scope for improvements- **duplicate keys**

- A 3-way partitioning method



Scope for improvements- **duplicate keys**

- A 3-way partitioning method



Scope for improvements- **duplicate keys**

- A 3-way partitioning method

10	1	5	13	10	2	17	10	3	10	19	10

Pivot

Scope for improvements- **duplicate keys**

- A 3-way partitioning method

10	1	5	13	10	2	17	10	3	10	19	10

Pivot

Scope for improvements- **duplicate keys**

- A 3-way partitioning method

10	1	5	13	10	2	17	10	3	10	19	10

Stop moving from left, an element greater than pivot is found

Pivot

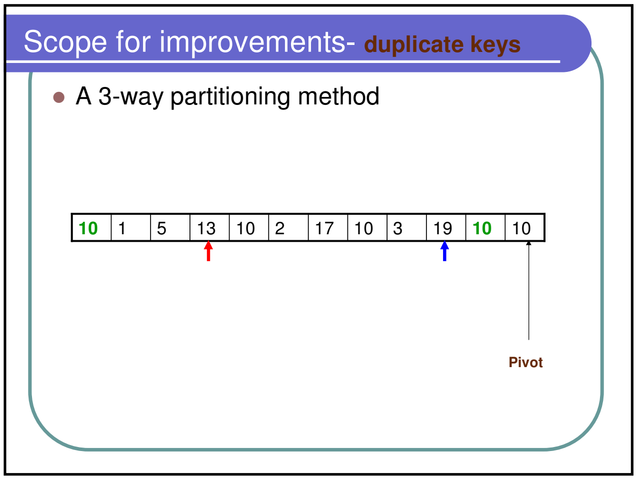
Scope for improvements- **duplicate keys**

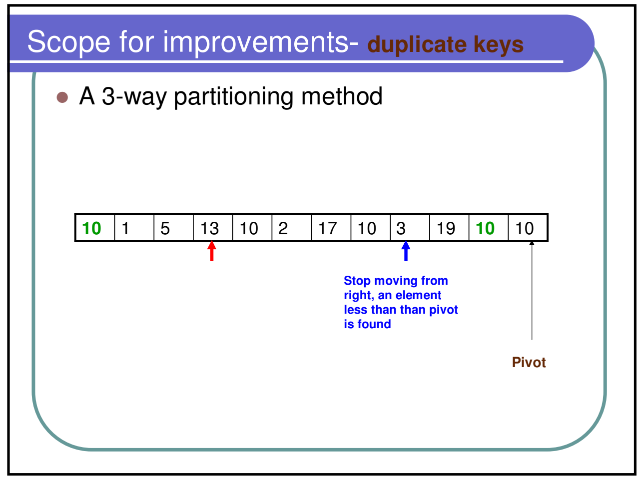
- A 3-way partitioning method

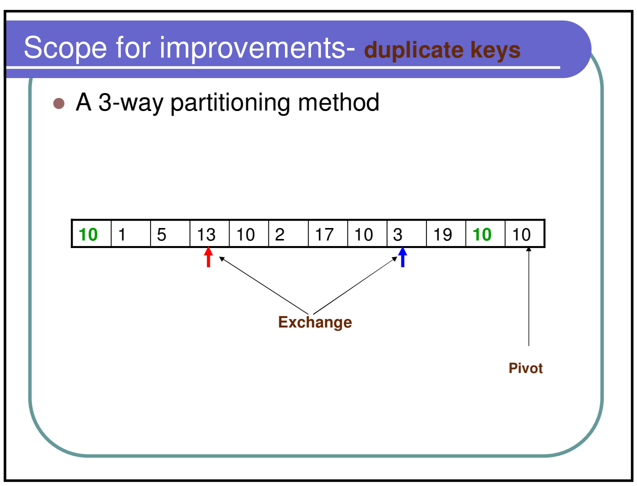
Equal to pivot, push to right

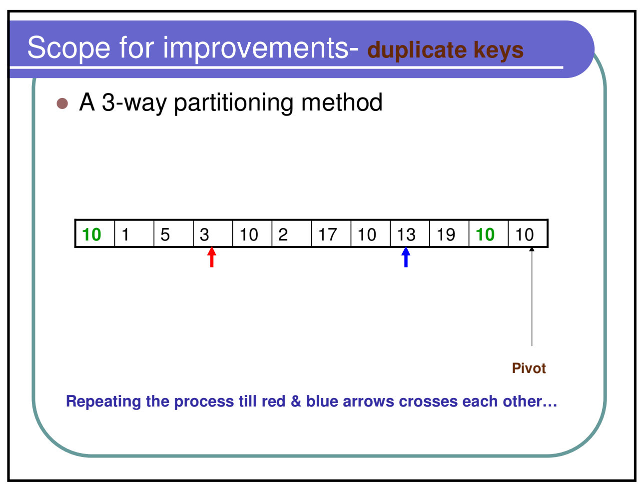
10	1	5	13	10	2	17	10	3	10	19	10

Pivot


Scope for improvements- duplicate keys
A 3-way partitioning method
10 1 5 13 10 2 17 10 3 19 10 10
Pivot

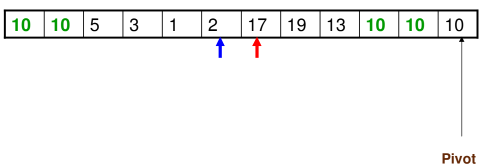

Scope for improvements- duplicate keys
A 3-way partitioning method
10 1 5 13 10 2 17 10 3 19 10 10
Stop moving from right, an element less than than pivot is found
Pivot


Scope for improvements- duplicate keys
A 3-way partitioning method
10 1 5 13 10 2 17 10 3 19 10 10
Exchange
Pivot


Scope for improvements- duplicate keys
A 3-way partitioning method
10 1 5 3 10 2 17 10 13 19 10 10
Pivot
Repeating the process till red & blue arrows crosses each other…

Scope for improvements- **duplicate keys**

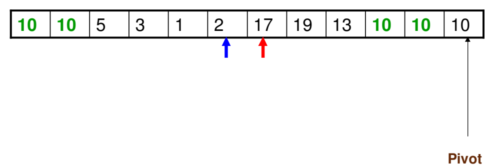
- A 3-way partitioning method



We reach here.........

Scope for improvements- **duplicate keys**

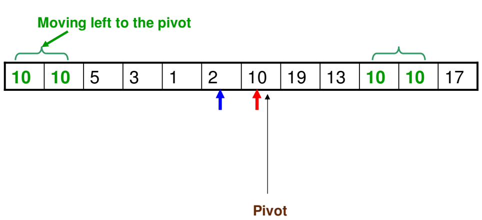
- A 3-way partitioning method



Exchange the pivot with red arrow content, we get…

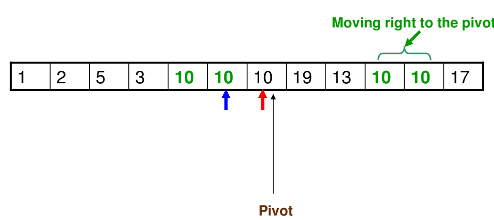
Scope for improvements- **duplicate keys**

- A 3-way partitioning method



Scope for improvements- **duplicate keys**

- A 3-way partitioning method



Scope for improvements- **duplicate keys**

- A 3-way partitioning method

Partition- 1				Partition- 2					Partition- 3		
1	2	5	3	**10**	**10**	10	**10**	**10**	19	13	17

Scope for improvements- **duplicate keys**

- A 3-way partitioning method

Partition- 1				Partition- 2					Partition- 3		
1	2	5	3	**10**	**10**	10	**10**	**10**	19	13	17

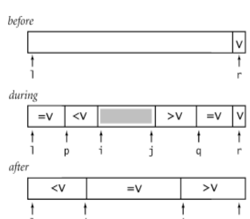
- **Apply Quick sort to partition-1 and partition-3, recursively……**
- **What if all the elements are same in the given array??????????**
- **Try to implement it….**

Implementation solution

3-way partitioning (Bentley-McIlroy): Partition elements into 4 parts:

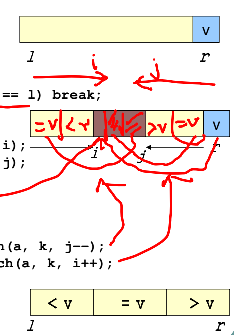
- no larger elements to left of i
- no smaller elements to right of j
- equal elements to left of p
- equal elements to right of q

Afterwards, swap equal keys into center.



Code

```
void sort(int a[], int l, int r) {
   if (r <= l) return;
   int i = l-1, j = r;
   int p = l-1, q = r;
   while(1)     {
        while (a[++i] < a[r]));
        while (a[r] < a[--j])) if (j == l) break;
        if (i >= j) break;
        exch(a, i, j);
        if (a[i]==a[r]) exch(a, ++p, i);
        if (a[j]==a[r]) exch(a, --q, j);
   }
   exch(a, i, r);
   j = i - 1;
   i = i + 1;
   for (int k = l ; k <= p; k++) exch(a, k, j--);
   for (int k = r-1; k >= q; k--) exch(a, k, i++);
   sort(a, l, j);
   sort(a, i, r);
}
```



Demo

- demo-partition3.ppt

Quiz 1

- Write two quick sort algorithms
  - 2-way partitioning
  - 3-way partitioning
- Create two identical arrays of 1 millions randomized numbers having value from 1 to 10.
- Compare the time for sorting the numbers using each algorithm

Guide

- Fill an array by random numbers

```
const int TOPITEM = 1000000;
void fill_array(void) {
        int i;
        float r;

        srand(time(NULL));

        for (i = 1; i < TOPITEM; i++) {
          r = (float) rand() / (float) RAND_MAX;
          data[i] = r * RANGE + 1;
        }
}
```

Demand memory

- For 1000000 elements
- int *w=(int *)malloc(1000000);

CPU Time Inquiry

```
#include <time.h>

   clock_t start, end;
   double cpu_time_used;

   start = clock();
   ... /* Do the work. */
   end = clock();
   cpu_time_used = ((double) (end - start)) /
  CLOCKS_PER_SEC;
```

Generalized sorting

- In C we can use the qsort function for sorting

```
void qsort(
      void *buf,
      size_t num,
      size_t size,
      int (*compare)(void const *, void  const *)
);
```

- The qsort() function sorts *buf* (which contains *num* items, each of size *size*).
- The *compare* function is used to compare the items in *buf*. *compare* should return negative if the first argument is less than the second, zero if they are equal, and positive if the first argument is greater than the second.

Example

```
int int_compare(void const* x, void const *y) {
  int m, n;
  m = *((int*)x);
  n = *((int*)y);
  if ( m == n ) return 0;
  return m > n ? 1: -1;
}
void main()
{
  int a[20], n;
  /* input an array of numbers */
  /* call qsort */
  qsort(a, n, sizeof(int), int_compare);
}
```

Function pointer

- Declare a pointer to a function
  - int (*pf) (int);
- Declare a function
  - int f(int);
- Assign a function to a function pointer
  - pf = &f;
- Call a function via pointer
  - ans = pf(5); // which are equivalent with ans = f(5)

- In the qsort() function, *compare* is a function pointer to reference to a compare the items

Quiz 2

- Write a function to compare strings so that it can be used with qsort() function
- Write a program to input a list of names, then use qsort() to sort this list and display the result.

Solution

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int cstring_cmp(const void *a, const void *b)
{
   const char **ia = (const char **)a;
   const char **ib = (const char **)b;
   return strcmp(*ia, *ib);
}

void print_cstring_array(char **array, size_t len)
{
   size_t i;

   for(i=0; i<len; i++)
      printf("%s | ", array[i]);

   putchar('\n');
}
```

Solution

```
int main()
{
   char *strings[] = { "Zorro", "Alex", "Celine", "Bill", "Forest", "Dexter" };
   size_t strings_len = sizeof(strings) / sizeof(char *);

   puts("*** String sorting...");

   print_cstring_array(strings, strings_len);

   qsort(strings, strings_len, sizeof(char *), cstring_cmp);

   print_cstring_array(strings, strings_len);

   return 0;
}
```

Solution: You can get strings from input also

```
int main()
{
   char strings[20];
   char *strings_array[20];
   int i = 0;
   int n;

   printf("\n Number of strings to sort:"); scanf("%d",&n);
   fflush(stdin);
   while(i<n){
         gets(strings);
         strings_array[i++] = strdup(strings);
   }
   print_cstring_array(strings_array, n);
   puts("*** String sorting...");
   qsort(strings_array, n, sizeof(char *), cstring_cmp);
   print_cstring_array(strings_array, n);
   return 0;
}
```

Quiz 3: Using qsort with array of structure

- Create an array of records, each record is in type of:

```
struct st_ex {
    char product[16];
    float price;
};
```

- Write a program using qsort to sort this array by the price and by product names.

Solution

- Create on your own function to compare two float numbers

```
int struct_cmp_by_price(const void *a, const void
   *b)
{
    struct st_ex *ia = (struct st_ex *)a;
    struct st_ex *ib = (struct st_ex *)b;
    return (int)(100.f*ia->price - 100.f*ib->price);
}
```

Solution

And by product names

```
int struct_cmp_by_product(const void *a, const
   void *b)
{
    struct st_ex *ia = (struct st_ex *)a;
    struct st_ex *ib = (struct st_ex *)b;
    return strcmp(ia->product, ib->product);
}
```

Solution: function for Output

```
void print_struct_array(struct st_ex *array, size_t
   len)
{
    size_t i;

    for(i=0; i<len; i++)
        printf("[ product: %s \t price: $%.2f ]\n",
   array[i].product, array[i].price);

    puts("--");
}
```

Solution: And test

```
void main()
{
    struct st_ex structs[] = {{"mp3 player", 299.0f}, {"plasma tv", 2200.0f},
                              {"notebook", 1300.0f}, {"smartphone", 499.99f},
                              {"dvd player", 150.0f}, {"matches", 0.2f }};

    size_t structs_len = sizeof(structs) / sizeof(struct st_ex);

    puts("*** Struct sorting (price)...");
    print_struct_array(structs, structs_len);

    qsort(structs, structs_len, sizeof(struct st_ex), struct_cmp_by_price);
    print_struct_array(structs, structs_len);
    puts("*** Struct sorting (product)...");
    qsort(structs, structs_len, sizeof(struct st_ex), struct_cmp_by_product);
    print_struct_array(structs, structs_len);
}
```

Quiz 4

- How to use qsort() to sort an array in descendant order?
- Write your own generalized quick sort function (using 3-way partitioning algorithm).
- Then, use this function to sort different kinds of data (integer numbers, phone number records, etc.)

Generalized sorting

- We can use also heap sort and merge sort

```
void heapsort(
      void *buf,
      size_t num,
      size_t size,
      int (*compare)(void const *, void  const *)
);


void mergesort(
      void *buf,
      size_t num,
      size_t size,
      int (*compare)(void const *, void  const *)
);
```

Exercise

- Using the grade data file of your class last semester.
- You write a compare function that takes the pointers to struct of student as parameters to use qsort to sort the student list.
- Change from qsort to heapsort