Intermediate Programming

— Sorting —

Waseda Univ.

Today's Topics

- Introduction to Sorting (implementation of sorting algorithms)
  - Bubble Sort
  - Quicksort
  - C library function - qsort()

Sorting

What is Sorting?

- Sorting arranges data in a certain order (ex. in ascending or descending order).
- Ordering
  - For two elements x, y, the odering is a binary relation that describes a rule of the arrangement.
  - It is denoted by $x < y$, $x = y$, or $x > y$ (the same as the numerical order).
  - No contradiction is admited such as $x < y$, $y < z$, and $z < x$.
  - $x = y$ denotes $x \leq y$ and $x \geq y$. In this case, the stable sorting algorithm preserves their relative order of the original list. The unstable sorting algorithm doesn't so.
  - The string can be sorted in a certain order (ex. the alphabetical order).
- Computational complexity is described by the big O notations with respect to the number of elements N. For typical sorting algorithms it is denoted by $\mathcal{O}(N^2)$, $\mathcal{O}(N \log N)$, and $\mathcal{O}(N)$.

Sorting algorithms

- Typical sorting algorithms are
  - Bubble sort
  - Insertion sort
  - Selection sort
  - Shell sort
  - Heapsort
  - Quicksort
  - Merge sort
  - Radix sort
  - etc.

Bubble sort

- Bubble sort is a simple sorting algorithm. It compares the first two elements, and swaps them. It continues doing this for each pair to the end of data.
- Bubble sort is a stable sort algorithm.
- The performance of this algorithm is $\mathcal{O}(N^2)$, so it is rarely used to sort large, unordered data. Bubble sort can be used for sorting a small number of items or for sorting a list that is nearly sorted.

Algorithm

For N elements we sort them in ascending order.

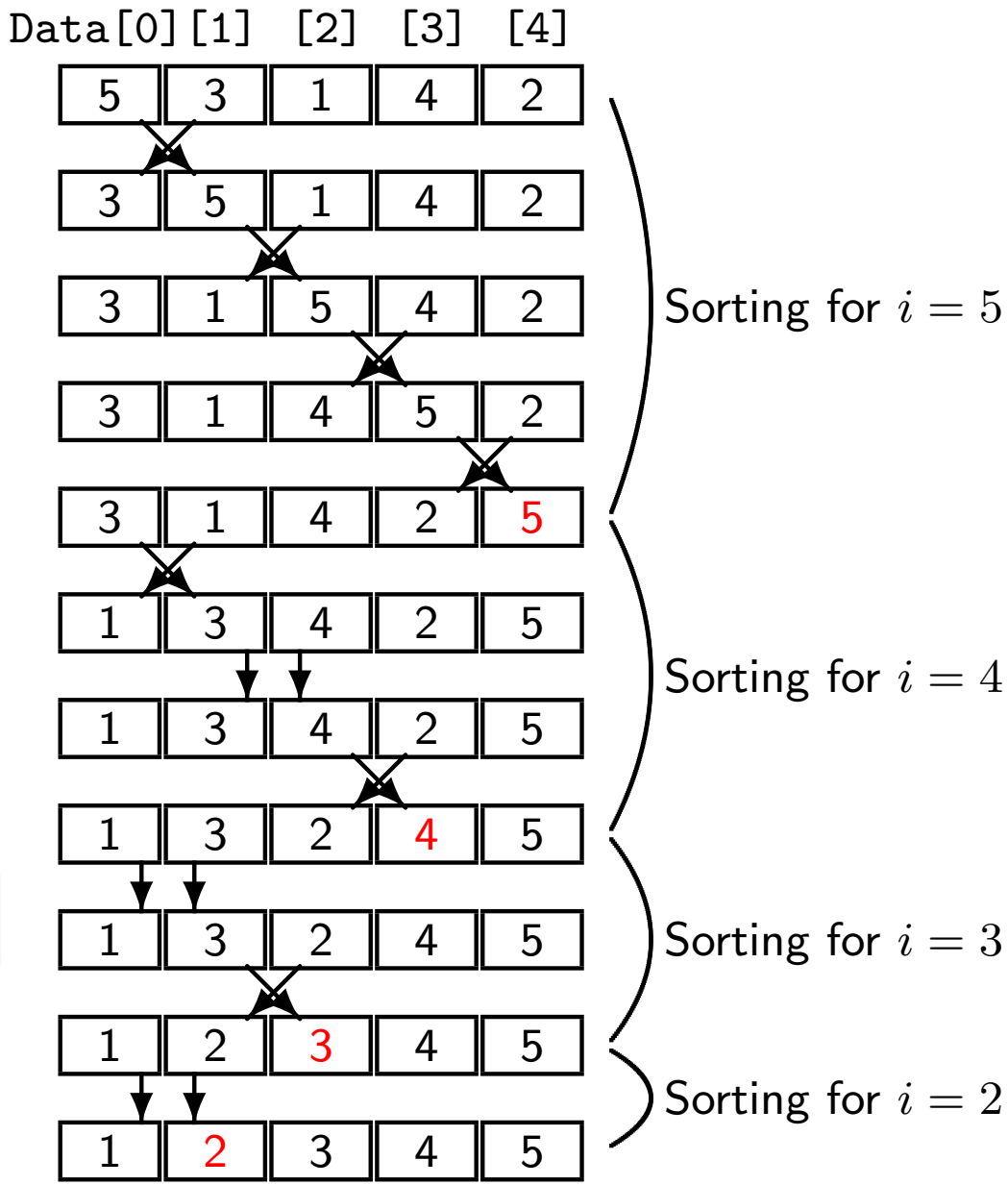
Step 1. For N, $N-1$,..., 2 elements, it continues the operation in Step 2.

Step 2. Compare the first two elements, and swap them if the first is greater than the second. Continue this for each pair to the end of elements.

Bubble sort

- Example for $N = 5$
- First we start sorting for $i = 5$ elements.
  Compare the two elements (index 0 and 1) and swap them. Compare the two elements (index 1 and 2) and swap them... Compare the two elements (index 3 and 4) and swap them.
- Then we start the sorting for $i = 4$ elements.
- Continue until $i = 2$.

Implementation is given by the above operations.



Exercise

Exercise: bubblesort.c

Let an array of `double` type be given by 5, 3, 1, 4, 2.
Write a program that arranges this data in ascending order by Bubble sort.

- Output of this program is as follows:

```
5, 3, 1, 4, 2,
3, 5, 1, 4, 2,
3, 1, 5, 4, 2,
3, 1, 4, 5, 2,
3, 1, 4, 2, 5,
1, 3, 4, 2, 5,
1, 3, 4, 2, 5,
1, 3, 2, 4, 5,
1, 3, 2, 4, 5,
1, 2, 3, 4, 5,
1, 2, 3, 4, 5,
```

Hint

A sample of this program is based on the following:
(Note that the size of array N can be calculated.)

```
#include <stdio.h>

void PrintData(double *Data, int N) {
  ...
}
void swap(double *x, double *y) {
  ...
}
void BubbleSort(double *Data, int N) {
  ...
}
int main(void) {
  double Data[] = {                    /* Number of elements are unknown */
    5, 3, 1, 4, 2
  };
  int N = sizeof(Data) / sizeof(Data[0]); /* The size of array */
  PrintData(Data, N);
  BubbleSort(Data, N);
  return 0;
}
```

sizeof() operator

`sizeof(type)` returns size in bytes of the object representation of type.

- Example
  - `sizeof(char)` represents 1.
  - `sizeof(int)` represents 4.
  - `sizeof(double)` represents 8.
- When we declare the following:

```
int i;
int A[5];
```

  it retuns
  - `sizeof(i)` $= 4$,
  - `sizeof(A[0])` $= 4$,
  - `sizeof(A)` $= 20$.
- `sizeof(A)/sizeof(A[0])` presents the size of the array A.

Answer

```
#include <stdio.h>

void PrintData(double *Data, int N) {
  /* Print each element of Data */
  int i;
  for(i=0;i<N;i++){
    printf("%.0f,",Data[i]);
  }
  printf("\n");
}

void swap(double *x, double *y) {
  /* Swap (*x) with (*y) */
  double z;
  z = *x; *x = *y; *y = z;
}
```

```
void BubbleSort(double *Data, int N) {
  /* Sorting Data in ascending order */
  int i,j;
  for(j=N;j>=2;j--){
    for(i=0;i<j-1;i++){
      if(Data[i]>Data[i+1]){
        swap(&Data[i],&Data[i+1]);
      }
      PrintData(Data,N);
    }
  }
}

int main(void) {
  double Data[] = {
    5, 3, 1, 4, 2
  };
  int N = sizeof(Data) / sizeof(Data[0]);
  PrintData(Data, N);
  BubbleSort(Data, N);
  return 0;
}
```

Answer

```c
#include <stdio.h>

void PrintData(double *Data, int N) {
  /* Print each element of Data */
  int i;
  for(i=0;i<N;i++){
    printf("%.0f,",Data[i]);
  }
  printf("\n");
}

void swap(double *x, double *y) {
  /* Swap (*x) with (*y) */
  double z;
  z = *x; *x = *y; *y = z;
}
```

```c
void BubbleSort(double *Data, int N) {
  /* Sorting Data in ascending order */
  int i,j;
  for(j=N;j>=2;j--){
    for(i=0;i<j-1;i++){
      if(Data[i]>Data[i+1]){
        swap(&Data[i],&Data[i+1]);
      }
      PrintData(Data,N);
    }
  }
}

int main(void) {
  double Data[] = {
    5, 3, 1, 4, 2
  };
  int N = sizeof(Data) / sizeof(Data[0]);
  PrintData(Data, N);
  BubbleSort(Data, N);
  return 0;
}
```

Answer

```
#include <stdio.h>

void PrintData(double *Data, int N) {
  /* Print each element of Data */
  int i;
  for(i=0;i<N;i++){
    printf("%.0f,",Data[i]);
  }
  printf("\n");
}

void swap(double *x, double *y) {
  /* Swap (*x) with (*y) */
  double z;
  z = *x; *x = *y; *y = z;
}
```

```
void BubbleSort(double *Data, int N) {
  /* Sorting Data in ascending order */
  int i,j;
  for(j=N;j>=2;j--){
    for(i=0;i<j-1;i++){
      if(Data[i]>Data[i+1]){
        swap(&Data[i],&Data[i+1]);
      }
      PrintData(Data,N);
    }
  }
}

int main(void) {
  double Data[] = {
    5, 3, 1, 4, 2
  };
  int N = sizeof(Data) / sizeof(Data[0]);
  PrintData(Data, N);
  BubbleSort(Data, N);
  return 0;
}
```

Answer

```c
#include <stdio.h>

void PrintData(double *Data, int N) {
  /* Print each element of Data */
  int i;
  for(i=0;i<N;i++){
    printf("%.0f,",Data[i]);
  }
  printf("\n");
}

void swap(double *x, double *y) {
  /* Swap (*x) with (*y) */
  double z;
  z = *x; *x = *y; *y = z;
}
```

```c
void BubbleSort(double *Data, int N) {
  /* Sorting Data in ascending order */
  int i,j;
  for(j=N;j>=2;j--){
    for(i=0;i<j-1;i++){
      if(Data[i]>Data[i+1]){
        swap(&Data[i],&Data[i+1]);
      }
      PrintData(Data,N);
    }
  }
}

int main(void) {
  double Data[] = {
    5, 3, 1, 4, 2
  };
  int N = sizeof(Data) / sizeof(Data[0]);
  PrintData(Data, N);
  BubbleSort(Data, N);
  return 0;
}
```

Quicksort

- Quicksort is a fast typical sorting algorithm based on a divide and conquer algorithm.
- Quicksort is an unstable sort algorithm.
- The average performance of this algorithm is $\mathcal{O}(N \log N)$ (the worst-case performance is $\mathcal{O}(N^2)$). This is a popular algorithm for sorting large data. Quicksort is available in many standard programming libraries.
- Quicksort relies on a partition:
  - An element called a pivot is selected in order to partition an array. The pivot is expected to be a median of the array. All elements smaller than the pivot are moved before it and all greater elements are moved after it.
  - The smaller and greater sub-arrays are then recursively sorted by applying the above step.
  - When the number of elements is less than 2, the algorithm stops for this sub-array.

Quicksort

Algorithm

For $x_L, x_{L+1}, ..., x_R$ we sort them in ascending order.

Step 1. Let $l = L$, $r = R$, and the pivot is determined by $p = x_{(L+R)/2}$.

Step 2. If $l > r$, go to Step 8.

Step 3. If $x_l < p$, assign $l \leftarrow l + 1$ and go to Step 3.

Step 4. If $x_r > p$, assing $r \leftarrow r - 1$ and go to Step 4.

Step 5. If $l > r$, go to Step 8.

Step 6. If $l < r$, swap x_l with x_r.

Step 7. Assign $l \leftarrow l + 1$ and $r \leftarrow r - 1$, back to Step 2.

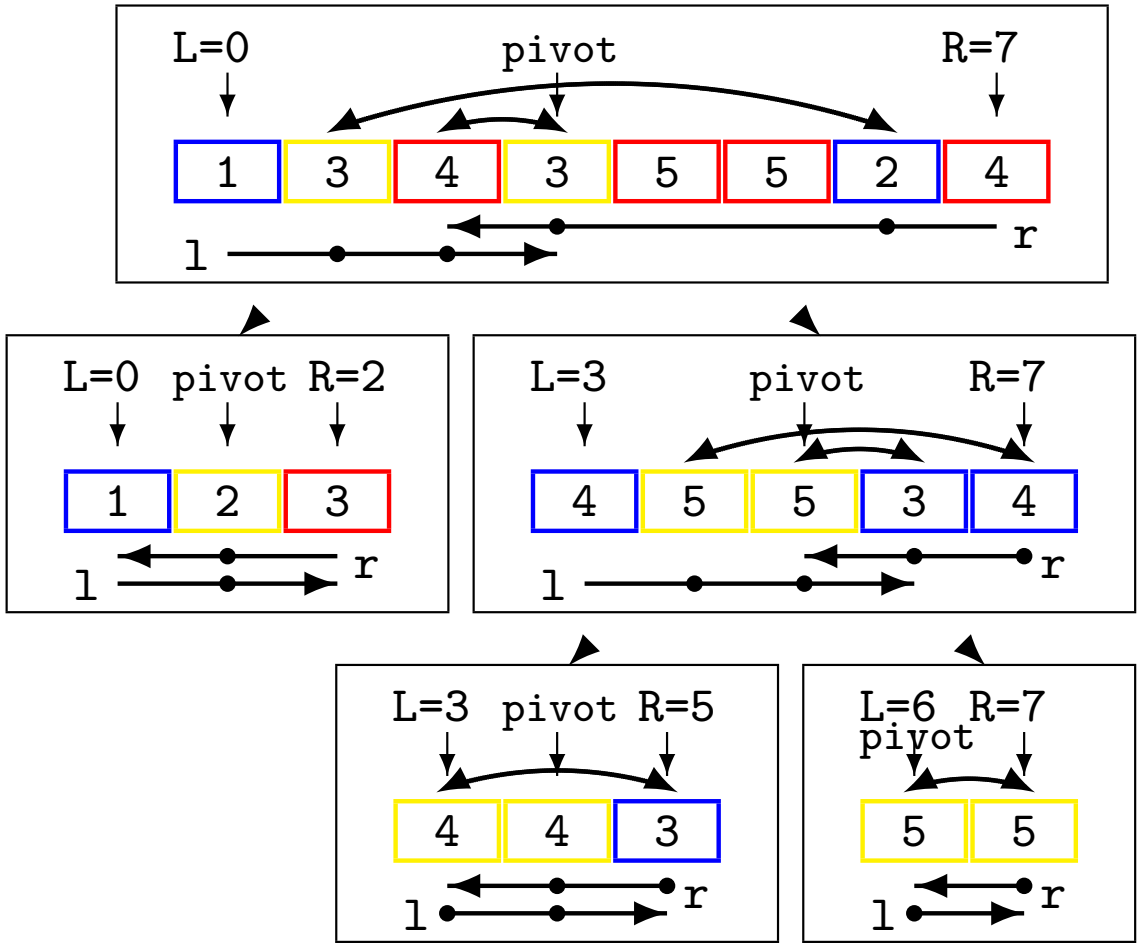
Step 8. If $L < r$, recursively apply this algorithm for $x_L, x_{L+1}, ..., x_r$.

Step 9. If $l < R$, recursively apply this algorithm for $x_l, x_{l+1}, ..., x_R$.

- Step 8, 9 implies that the number of elements is greater or equal to 2.
- First we apply this algorithm for $L = 0$ and $R = N-1$ in order to sort the whole array (N elements).

Quicksort

- The pivot is 3.
  Blue: smaller element,
  Red: greater element,
  Yellow: same value.
- "l" finds a greater or same element from the left side. "r" finds a smaller or same element from the right.
- If l<r, swap these elements.
- This process ends if l>r. Recursively apply this operation to smaller and greater sub-arrays.



C library function - qsort

Description of the qsort function

- ```
  void qsort(void *data,
             size_t num,
             size_t size,
             int (*func)(const void *, const void*))
  ```

- Include “stdlib.h”.
- data : The first pointer to the array “data” (to be sorted).
- size_t num : The number of elements in the array pointed by “data”.
- size_t size : The size in bytes of each element in the array.
- int func : This is the function that compares two elements.
  ```

Example for using qsort

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

int compare(const void *a, const void *b){ /* Define the compare function */
    return *(int*)a-*(int*)b;
}

int main(void){
    int i;
    int data[] = {6,3,5,4,2,1};
    int N = sizeof(data) / sizeof(data[0]);
    printf("Before sorting the list is:");
    for (i=0; i<N; i++){
        printf("%3d",data[i]);
    }
    printf("\n");
    printf("After sorting the list is:");
    qsort(data,N,sizeof(int),compare);
    for (i=0; i<N; i++){
        printf("%3d",data[i]);
    }
    printf("\n");
    return 0;
}
```

Example for using qsort

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

int compare(const void *a, const void *b){ /* Define the compare function */
    return *(int*)a-*(int*)b;
}

int main(void){
    int i;
    int data[] = {6,3,5,4,2,1};
    int N = sizeof(data) / sizeof(data[0]);
    printf("Before sorting the list is:");
    for (i=0; i<N; i++){
        printf("%3d",data[i]);
    }
    printf("\n");
    printf("After sorting the list is:");
    qsort(data,N,sizeof(int),compare);
    for (i=0; i<N; i++){
        printf("%3d",data[i]);
    }
    printf("\n");
    return 0;
}
```

Example for using qsort

```
int compare(const void *a, const void *b){
        return *(int*)a-*(int*)b;
}
```

- If $*a < *b$, return a negative value.
- If $*a = *b$, return 0.
- If $*a > *b$, return a positive value.
- Sorting is done by the above ordering function.

```
qsort(data,N,sizeof(int),compare);
```

- data : Name of an array to be sorted.
- The number of elements are given by N.
- The "sizeof(int)" returns the size in bytes for "int" (4bytes).
- The "compare" is the name of the ordering function.

Summary

- Introduction to Sorting (implementation of sorting algorithms)
  - Bubble Sort
  - Quicksort
  - C library function - qsort()