

ECE 264 Advanced C Programming

2009/04/29

Contents

1 Quicksort	1
-------------	---

1 Quicksort

If we can sort numbers using a binary search tree in $n \lg n$ steps, can we sort as fast, or even faster, using an array? Remember that in an array, we can retrieve any element in a single step. In a binary tree (or a linked list), we have to visit nodes one by one. There are many ways to sort numbers in an array using $n \lg n$ steps. Here, we introduce a method called *quicksort*.

```
#include <stdio.h>

void swap(int *a, int *b)
{
    int t = *a;
    *a = *b;
    *b = t;
}

void printArray(int arr[], int size)
{
    int i;
    for (i = 0; i < size; i++)
    {
        printf("%d ", arr[i]);
    }
    printf("\n\n");
}

void sort(int arr[], int first, int last, int size)
{
    if (last > first + 1)
    {
```

```

int pivot = arr[first];
printf("first = %d, last = %d, pivot = %d\n",
       first, last, pivot);
int left = first + 1;
int right = last;
while (left < right)
{
    if (arr[left] <= pivot)
        { left++; }
    else
        { swap(&arr[left], &arr[--right]); }
}
swap(&arr[--left], &arr[first]);
printArray(arr, size);
sort(arr, first, left, size);
sort(arr, right, last, size);
}

int main(int argc, char * argv[])
{
    int data[] = {5, 4, 3, 9, -1, 0, 4, 3, 2, 11, 7, 6, 8, 9, 14};
    int length = sizeof(data)/ sizeof(int);
    int index;
    printArray(data, length);
    sort(data, 0, length, length);
    for (index = 0; index < length; index++)
    {
        printf("%d ", data[index]);
    }
    printf("\n");
    return 0;
}

/*
5 4 3 9 -1 0 4 3 2 11 7 6 8 9 14

first = 0, last = 15, pivot = 5
3 4 3 2 -1 0 4 5 11 7 6 8 9 14 9

first = 0, last = 7, pivot = 3

```

```

-1 0 3 2 3 4 4 5 11 7 6 8 9 14 9

first = 0, last = 4, pivot = -1
-1 3 2 0 3 4 4 5 11 7 6 8 9 14 9

first = 1, last = 4, pivot = 3
-1 0 2 3 3 4 4 5 11 7 6 8 9 14 9

first = 1, last = 3, pivot = 0
-1 0 2 3 3 4 4 5 11 7 6 8 9 14 9

first = 5, last = 7, pivot = 4
-1 0 2 3 3 4 4 5 11 7 6 8 9 14 9

first = 8, last = 15, pivot = 11
-1 0 2 3 3 4 4 5 9 7 6 8 9 11 14

first = 8, last = 13, pivot = 9
-1 0 2 3 3 4 4 5 9 7 6 8 9 11 14

first = 8, last = 12, pivot = 9
-1 0 2 3 3 4 4 5 8 7 6 9 9 11 14

first = 8, last = 11, pivot = 8
-1 0 2 3 3 4 4 5 6 7 8 9 9 11 14

first = 8, last = 10, pivot = 6
-1 0 2 3 3 4 4 5 6 7 8 9 9 11 14

-1 0 2 3 3 4 4 5 6 7 8 9 9 11 14

*/

```

The basic concept is the same: avoid redundant work. If two numbers a and b ($a > b$) have been compared. When we encounter c and $c > a$, we know $c > b$ and it is unnecessary to compare b and c .

C has a built-in function for quicksort. In the lecture for March 11, we already used it:

```

#include <time.h>
#include <stdio.h>
#include <stdlib.h>
int search1(int data[], int size, int value);

```

```

int search2(int data[], int size, int value);

void print(int data[], int size)
{
    int ind;
    for (ind = 0; ind < size; ind++)
    {
        printf("%d ", data[ind]);
    }
    printf("\n");
}

int compare(const void * p1, const void * p2)
/* comparison function needed by qsort */
{
    int v1 = * (const int *) p1;
    int v2 = * (const int *) p2;
    if (v1 > v2) { return 1; }
    if (v1 < v2) { return -1; }
    return 0;
}

int main(int argc, char * argv[])
{
    int numElement = 0;
    srand(time(0));
    if (argc > 1)
    {
        numElement = (int)strtol(argv[1], (char **)NULL, 10);
    }
    if (numElement < 100) { numElement = 100; }
    int * data = malloc(numElement * sizeof(int));
    int ind;
    /* initialize the elements */
    for (ind = 0; ind < numElement; ind++)
    {
        data[ind] = numElement - ind;
    }
    print(data, numElement);
    int value = rand() % (5 * numElement);
    printf("search %d, index = %d\n", value,
           search1(data, numElement, value));
}

```

```
qsort(data, numElement, sizeof(int), compare);
/* qsort (quicksort) is provided by C but we have to provide a
   comparison function. Please check the manual for details */
print(data, numElement);
printf("search %d, index = %d\n", value,
       search2(data, numElement, value));
/* The index may be different from search1 because the elements have
   been sorted. */
free (data);
return 0;
}
```

To use C's quicksort function, we have to provide a compare function. This allows us to use quicksort for sorting structures.