Sorting Lesson Outline

How to Sort an Array?

-23	97	18	21	5	-86	64	0	-37

Suppose you have a big array full of data, and you want to put it into order, either from lowest to highest (ascending) or highest to lowest (descending).

How can you sort it so that it's in the order you want?

Many Sorting Algorithms

There are many sorting algorithms, for example:

- Bubble Sort
- Selection Sort
- Quick Sort

… and many more.

Bubble Sort #1

```
#include <stdio.h>

int main ()
{ /* main */
    const int minimum_length       =  1;
    const int first_element        =  0;
    const int program_failure_code = -1;
    const int program_success_code =  0;
    int* array = (int*)NULL;
    int length, element;
    void bubble_sort(int* array, int length);
    void print_array(int* array, int length);
```

http://en.wikipedia.org/wiki/Bubble_sort

Bubble Sort #2

```
    printf("I'm going to sort an array for you.\n");
    printf("How many elements will the array have?\n");
    scanf("%d", &length);
    if (length < minimum_length) {
        printf("ERROR: can't have an array of length %d.\n",
            length);
        exit(program_failure_code);
    } /* if (length < minimum_length) */
    array = (int*)malloc(sizeof(int) * length);
    if (array == (int*)NULL) {
        printf("ERROR: can't allocate an array of length %d.\n",
            length);
        exit(program_failure_code);
    } /* if (array == (int*)NULL) */
    printf("What are the %d elements, unsorted?\n", length);
    for (element = first_element; element < length; element++) {
        scanf("%d", &array[element]);
    } /* for element */
    bubble_sort(array, length);
    printf("The %d elements, sorted, are:\n", length);
    print_array(array, length);
    return program_success_code;
} /* main */
```

Bubble Sort #3

```
void bubble_sort (int* array, int length)
{ /* bubble_sort */
    const int  first_element   = 0;
    const int  first_iteration = 0;
    const char false           = 0;
    const char true            = 1;
    int  temporary;
    int  element;
    int  iteration;
    char this_iteration_had_a_swap; /* Boolean */
    void print_array(int* array, int length);
```

Bubble Sort #4

```
    iteration = first_iteration;
    this_iteration_had_a_swap = true;
    while (this_iteration_had_a_swap) {
        printf("\n");
        this_iteration_had_a_swap = false;
        for (element = first_element;
             element < (length - 1); element++) {
            if (array[element] > array[element + 1]) {

                /* Swap! */
                temporary           = array[element];
                array[element]      = array[element + 1];
                array[element + 1] = temporary;

                this_iteration_had_a_swap = true;
            } /* if (array[ element ] > array[ element + 1 ]) */
            printf("After iteration #%d", iteration);
            printf(" comparing elements #%d/%d:\n",
                element, element + 1);
            print_array(array, length);
        } /* for element */
        iteration++;
    } /* while (this_iteration_had_a_swap) */
} /* bubble_sort */
```

Bubble Sort #5

```
void print_array (int* array, int length)
{ /* print_array */
    const int  first_element =  0;
    int element;

    for (element = first_element; element < length; element++) {
        printf("%d ", array[element]);
    } /* for element */
    printf("\n");
} /* print_array */
```

Bubble Sort Run Start

```
% bubblesort
I'm going to sort an array for you.
How many elements will the array have?
9
What are the 9 elements, unsorted?
-23  97  18  21   5 -86  64   0 -37
```

Bubble Sort Run Iteration #1

```
After iteration #1 comparing elements #0/1:
-23  97  18  21   5 -86  64   0 -37
After iteration #1 comparing elements #1/2:
-23  18  97  21   5 -86  64   0 -37
After iteration #1 comparing elements #2/3:
-23  18  21  97   5 -86  64   0 -37
After iteration #1 comparing elements #3/4:
-23  18  21   5  97 -86  64   0 -37
After iteration #1 comparing elements #4/5:
-23  18  21   5 -86  97  64   0 -37
After iteration #1 comparing elements #5/6:
-23  18  21   5 -86  64  97   0 -37
After iteration #1 comparing elements #6/7:
-23  18  21   5 -86  64   0  97 -37
After iteration #1 comparing elements #7/8:
-23  18  21   5 -86  64   0 -37  97
```

Biggest!

Bubble Sort Run Iteration #2

```
After iteration #2 comparing elements #0/1:
-23  18  21   5 -86  64   0 -37  97
After iteration #2 comparing elements #1/2:
-23  18  21   5 -86  64   0 -37  97
After iteration #2 comparing elements #2/3:
-23  18   5  21 -86  64   0 -37  97
After iteration #2 comparing elements #3/4:
-23  18   5 -86  21  64   0 -37  97
After iteration #2 comparing elements #4/5:
-23  18   5 -86  21  64   0 -37  97
After iteration #2 comparing elements #5/6:
-23  18   5 -86  21   0  64 -37  97
After iteration #2 comparing elements #6/7:
-23  18   5 -86  21   0 -37  64  97
After iteration #2 comparing elements #7/8:
-23  18   5 -86  21   0 -37  64  97
```

Second biggest!

Bubble Sort Run Iteration #3

```
After iteration #3 comparing elements #0/1:
-23   18    5 -86   21    0 -37   64   97
After iteration #3 comparing elements #1/2:
-23    5   18 -86   21    0 -37   64   97
After iteration #3 comparing elements #2/3:
-23    5  -86  18   21    0 -37   64   97
After iteration #3 comparing elements #3/4:
-23    5 -86   18   21    0 -37   64   97
After iteration #3 comparing elements #4/5:
-23    5 -86   18    0   21 -37   64   97
After iteration #3 comparing elements #5/6:
-23    5 -86   18    0  -37   21   64   97
After iteration #3 comparing elements #6/7:
-23    5 -86   18    0 -37   21   64   97
After iteration #3 comparing elements #7/8:
-23    5 -86   18    0 -37   21   64   97
```

Bubble Sort Run Iteration #4

```
After iteration #4 comparing elements #0/1:
-23   5 -86  18   0 -37  21  64  97
After iteration #4 comparing elements #1/2:
-23 -86   5  18   0 -37  21  64  97
After iteration #4 comparing elements #2/3:
-23 -86   5  18   0 -37  21  64  97
After iteration #4 comparing elements #3/4:
-23 -86   5   0  18 -37  21  64  97
After iteration #4 comparing elements #4/5:
-23 -86   5   0 -37  18  21  64  97
After iteration #4 comparing elements #5/6:
-23 -86   5   0 -37  18  21  64  97
After iteration #4 comparing elements #6/7:
-23 -86   5   0 -37  18  21  64  97
After iteration #4 comparing elements #7/8:
-23 -86   5   0 -37  18  21  64  97
```

Bubble Sort Run Iteration #5

```
After iteration #5 comparing elements #0/1:
-86 -23   5   0 -37  18  21  64  97
After iteration #5 comparing elements #1/2:
-86 -23   5   0 -37  18  21  64  97
After iteration #5 comparing elements #2/3:
-86 -23   0   5 -37  18  21  64  97
After iteration #5 comparing elements #3/4:
-86 -23   0 -37   5  18  21  64  97
After iteration #5 comparing elements #4/5:
-86 -23   0 -37   5  18  21  64  97
After iteration #5 comparing elements #5/6:
-86 -23   0 -37   5  18  21  64  97
After iteration #5 comparing elements #6/7:
-86 -23   0 -37   5  18  21  64  97
After iteration #5 comparing elements #7/8:
-86 -23   0 -37   5  18  21  64  97
```

Bubble Sort Run Iteration #6

```
After iteration #6 comparing elements #0/1:
-86 -23   0 -37   5  18  21  64  97
After iteration #6 comparing elements #1/2:
-86 -23   0 -37   5  18  21  64  97
After iteration #6 comparing elements #2/3:
-86 -23 -37   0   5  18  21  64  97
After iteration #6 comparing elements #3/4:
-86 -23 -37   0   5  18  21  64  97
After iteration #6 comparing elements #4/5:
-86 -23 -37   0   5  18  21  64  97
After iteration #6 comparing elements #5/6:
-86 -23 -37   0   5  18  21  64  97
After iteration #6 comparing elements #6/7:
-86 -23 -37   0   5  18  21  64  97
After iteration #6 comparing elements #7/8:
-86 -23 -37   0   5  18  21  64  97
```

Bubble Sort Run Iteration #7

```
After iteration #7 comparing elements #0/1:
-86 -23 -37   0   5  18  21  64  97
After iteration #7 comparing elements #1/2:
-86 -37 -23   0   5  18  21  64  97
After iteration #7 comparing elements #2/3:
-86 -37 -23   0   5  18  21  64  97
After iteration #7 comparing elements #3/4:
-86 -37 -23   0   5  18  21  64  97
After iteration #7 comparing elements #4/5:
-86 -37 -23   0   5  18  21  64  97
After iteration #7 comparing elements #5/6:
-86 -37 -23   0   5  18  21  64  97
After iteration #7 comparing elements #6/7:
-86 -37 -23   0   5  18  21  64  97
After iteration #7 comparing elements #7/8:
-86 -37 -23   0   5  18  21  64  97
```

Bubble Sort Run Iteration #8

```
After iteration #8 comparing elements #0/1:
-86 -37 -23   0   5  18  21  64  97
After iteration #8 comparing elements #1/2:
-86 -37 -23   0   5  18  21  64  97
After iteration #8 comparing elements #2/3:
-86 -37 -23   0   5  18  21  64  97
After iteration #8 comparing elements #3/4:
-86 -37 -23   0   5  18  21  64  97
After iteration #8 comparing elements #4/5:
-86 -37 -23   0   5  18  21  64  97
After iteration #8 comparing elements #5/6:
-86 -37 -23   0   5  18  21  64  97
After iteration #8 comparing elements #6/7:
-86 -37 -23   0   5  18  21  64  97
After iteration #8 comparing elements #7/8:
-86 -37 -23   0   5  18  21  64  97
```

Nothing got swapped, so done!

Bubble Sort Run End

```
The 9 elements, sorted, are:
-86 -37 -23   0   5  18  21  64  97
```

Time Complexity of Bubble Sort #1

```
    iteration = first_iteration;
    this_iteration_had_a_swap = true;
    while (this_iteration_had_a_swap) {
        printf("\n");
        this_iteration_had_a_swap = false;
        for (element = first_element;
             element < (length - 1); element++) {
            if (array[element] > array[element + 1]) {

                /* Swap! */
                temporary           = array[element];
                array[element]      = array[element + 1];
                array[element + 1] = temporary;

                this_iteration_had_a_swap = true;
            } /* if (array[ element ] > array[ element + 1 ]) */
            printf("After iteration #%d", iteration);
            printf(" comparing elements #%d/%d:\n",
                element, element + 1);
            print_array(array, length);
        } /* for element */
        iteration++;
    } /* while (this_iteration_had_a_swap) */
} /* bubble_sort */
```

Time Complexity of Bubble Sort #2

What is the time complexity of bubble sort?

Suppose the array has length *n*.

Well, the inner `for` loop always has *n*-1 iterations.

How many iterations does the outer `while` loop have?

Well, the outer loop will iterate until the whole array is sorted (plus one extra iteration to determine that it is sorted).

On average, it'll have to do that about ½ *n* times.

Therefore, the time complexity is **O**(n^2) – which is **BAD**.

Selection Sort

Selection sort works by selecting the lowest (or highest) value in the list and swapping it with the first (or last) element, then finding the second lowest (or highest) and swapping it with the second (or next-to-last) element, and so on.

http://en.wikipedia.org/wiki/Selection_sort

Selection Sort #1

```
void selection_sort (int* array, int length)
{ /* selection_sort */
    const int  first_element = 0;
    int temporary;
    int element1, element2, minimum_element;
    void print_array(int* array, int length);
```

Selection Sort #2

```
    for (element1 = first_element;
         element1 < (length - 1); element1++) {

        /* Find the minimum. */
        minimum_element = element1;
        for (element2 = element1 + 1;
             element2 < length; element2++) {
            if (array[element2] < array[minimum_element]) {
                minimum_element = element2;
            } /* if (array[element2] < array[minimum_element]) */
            printf("Element %d is %d, minimum element at %d is %d.\n",
                element2, array[element2],
                minimum_element, array[minimum_element]);

        } /* for element2 */
        /* Swap! */
        temporary               = array[element1];
        array[element1]         = array[minimum_element];
        array[minimum_element] = temporary;
        printf("Minimum #%d had index %d",
            element1 + 1, minimum_element);
        printf(" but now has index %d:\n", element1);
        print_array(array, length);
    } /* for element1 */
} /* selection_sort */
```

Selection Sort Run Start

```
I'm going to sort an array for you.
How many elements will the array have?
9
What are the 9 elements, unsorted?
-23  97  18  21   5 -86  64   0 -37
```

Selection Sort Run Minimum #1

```
-23  97  18  21   5 -86  64   0 -37
Element 1 is 97, minimum element at 0 is -23.
Element 2 is 18, minimum element at 0 is -23.
Element 3 is 21, minimum element at 0 is -23.
Element 4 is 5, minimum element at 0 is -23.
Element 5 is -86, minimum element at 5 is -86.
Element 6 is 64, minimum element at 5 is -86.
Element 7 is 0, minimum element at 5 is -86.
Element 8 is -37, minimum element at 5 is -86.
Minimum #1 had index 5 but now has index 0:
-86  97  18  21   5 -23  64   0 -37
```

Selection Sort Run Minimum #2

```
-86  97  18  21   5 -23  64   0 -37
Element 2 is 18, minimum element at 2 is 18.
Element 3 is 21, minimum element at 2 is 18.
Element 4 is 5, minimum element at 4 is 5.
Element 5 is -23, minimum element at 5 is -23.
Element 6 is 64, minimum element at 5 is -23.
Element 7 is 0, minimum element at 5 is -23.
Element 8 is -37, minimum element at 8 is -37.
Minimum #2 had index 8 but now has index 1:
-86 -37  18  21   5 -23  64   0  97
```

Selection Sort Run Minimum #3

```
-86 -37  18  21   5 -23  64   0  97
Element 3 is 21, minimum element at 2 is 18.
Element 4 is 5, minimum element at 4 is 5.
Element 5 is -23, minimum element at 5 is -23.
Element 6 is 64, minimum element at 5 is -23.
Element 7 is 0, minimum element at 5 is -23.
Element 8 is 97, minimum element at 5 is -23.
Minimum #3 had index 5 but now has index 2:
-86 -37 -23  21   5  18  64   0  97
```

Selection Sort Run Minimum #4

```
-86 -37 -23  21   5  18  64   0  97
Element 4 is 5, minimum element at 4 is 5.
Element 5 is 18, minimum element at 4 is 5.
Element 6 is 64, minimum element at 4 is 5.
Element 7 is 0, minimum element at 7 is 0.
Element 8 is 97, minimum element at 7 is 0.
Minimum #4 had index 7 but now has index 3:
-86 -37 -23   0   5  18  64  21  97
```

Selection Sort Run Minimum #5

```
-86 -37 -23   0   5  18  64  21  97
Element 5 is 18, minimum element at 4 is 5.
Element 6 is 64, minimum element at 4 is 5.
Element 7 is 21, minimum element at 4 is 5.
Element 8 is 97, minimum element at 4 is 5.
Minimum #5 had index 4 but now has index 4:
-86 -37 -23   0   5  18  64  21  97
```

Selection Sort Run Minimum #6

```
-86 -37 -23   0   5  18  64  21  97
Element 6 is 64, minimum element at 5 is 18.
Element 7 is 21, minimum element at 5 is 18.
Element 8 is 97, minimum element at 5 is 18.
Minimum #6 had index 5 but now has index 5:
-86 -37 -23   0   5  18  64  21  97
```

Selection Sort Run Minimum #7

```
-86 -37 -23   0   5  18  64  21  97
Element 7 is 21, minimum element at 7 is 21.
Element 8 is 97, minimum element at 7 is 21.
Minimum #7 had index 7 but now has index 6:
-86 -37 -23   0   5  18  21  64  97
```

Selection Sort Run Minimum #8

```
-86 -37 -23   0   5  18  21  64  97
Element 8 is 97, minimum element at 7 is 64.
Minimum #8 had index 7 but now has index 7:
-86 -37 -23   0   5  18  21  64  97
```

Selection Sort Run End

```
The 9 elements, sorted, are:
-86 -37 -23   0   5  18  21  64  97
```

Time Complexity of Selection Sort #1

```
    for (element1 = first_element;
         element1 < (length - 1); element1++) {

        /* Find the minimum. */
        minimum_element = element1;
        for (element2 = element1 + 1;
             element2 < length; element2++) {
            if (array[element2] < array[minimum_element]) {
                minimum_element = element2;
            } /* if (array[element2] < array[minimum_element]) */
        } /* for element2 */

        /* Swap! */
        temporary               = array[element1];
        array[element1]         = array[minimum_element];
        array[minimum_element] = temporary;

        printf("Minimum #%d had index %d",
            element1, minimum_element);
        printf(" but now has index %d:\n", element1);
        print_array(array, length);
    } /* for element1 */
} /* selection_sort */
```

Time Complexity of Selection Sort #2

What is the time complexity of selection sort?

Suppose the array has length n.

Well, the outer `for` loop always has n-1 iterations.

How many iterations does the inner `for` loop have?

Well, for the first iteration of the outer loop, the inner loop has n-1 iterations; for the second iteration of the outer loop, the inner loop has n-2 iterations; and so on.

On average, the inner loop will have approximately $\frac{1}{2}n$ iterations.

Therefore, the time complexity is $\mathbf{O}(n^2)$ – which is **BAD**.

Quick Sort #1

```
void quicksort (int* array, int length)
{ /* quicksort */
    void quicksort_helper(int* array, int length,
                          int low, int high);

    quicksort_helper(array, length, 0, length - 1);
} /* quicksort */
```

http://www.inf.fh-flensburg.de/lang/algorithmen/sortieren/quick/quicken.htm

Quick Sort #2

```
void quicksort_helper (int* array, int length, int low, int high)
{ /* quicksort_helper */
    int temporary;
    int element1, element2, middle;
    int pivot;
    void print_array(int* array, int length);

    element1 = low;
    element2 = high;
    middle   = (low + high) / 2;
    pivot    = array[middle];
```

Quick Sort #3

```
/* Partition this part of the array into halves. */
   do {
       while (array[element1] < pivot) {
           element1++;
           printf("low %d, high %d, pivot %d, element1 %d\n",
               low, high, pivot, element1);
           print_array(array, length);
       } /* while (array[element1] < pivot) */
       while (array[element2] > pivot) {
           element2--;
           printf("low %d, high %d, pivot %d, element2 %d\n",
               low, high, pivot, element2);
           print_array(array, length);
       } /* while (array[element2] > pivot) */
```

Quick Sort #4

```
    /* Now, all values at indices between element1
     * and element2 are in the wrong place. */
     if (element1 <= element2) {
         /* Swap! */
         temporary       = array[element1];
         array[element1] = array[element2];
         array[element2] = temporary;
         element1++;
         element2--;
         printf("low %d, high %d, pivot %d,",
             low, high, pivot);
         printf(" element1 %d, element2 %d:\n",
             element1, element2);
         print_array(array, length);
     } /* if (element1 <= element2) */
} while (element1 <= element2);
```

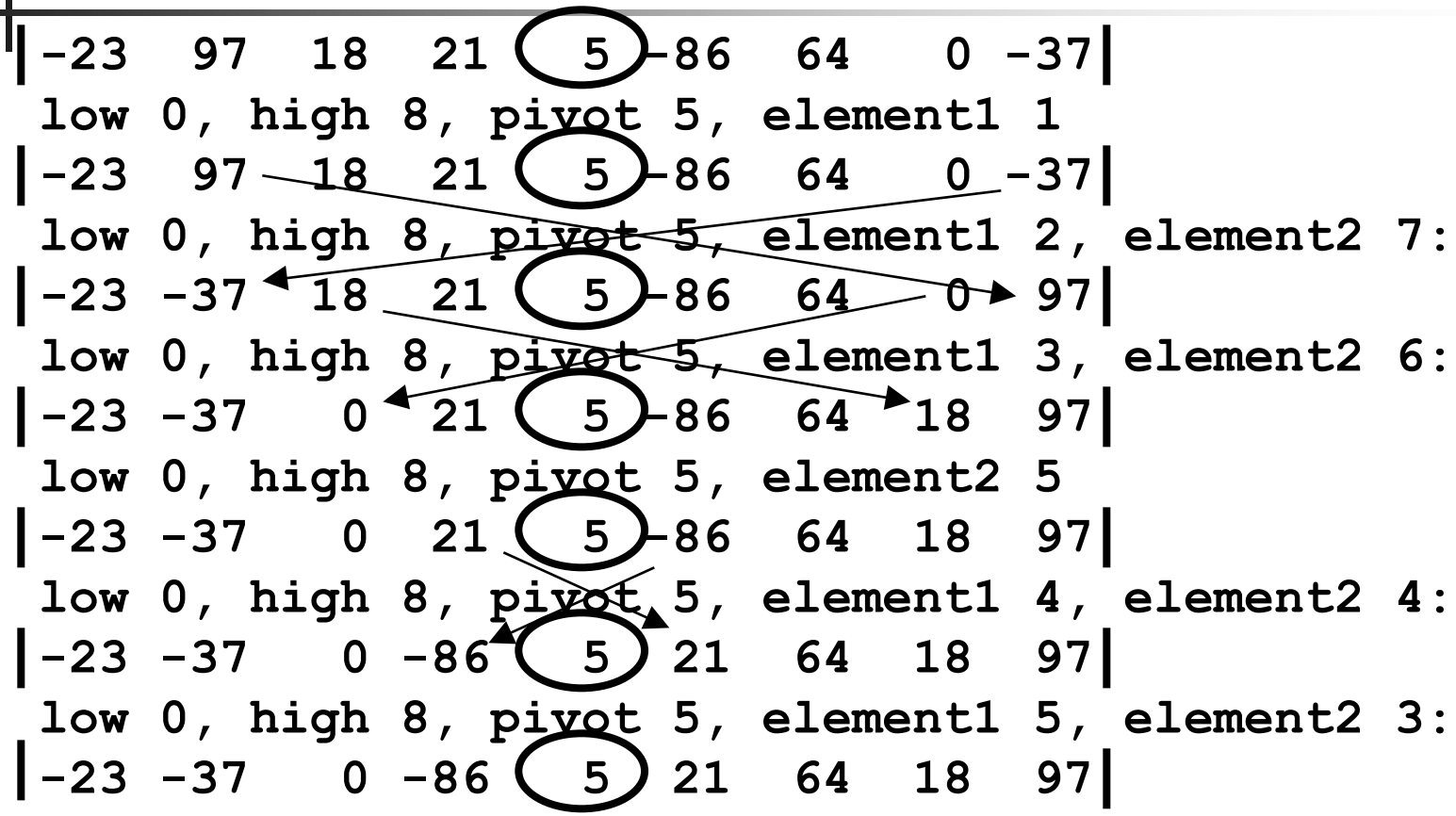
Quick Sort #5

```
    /* Recursively sort the two halves. */
     if (low < element2) {
         quicksort_helper(array, length, low, element2);
     } /* if (low < element2) */
     if (element1 < high) {
        quicksort_helper(array, length, element1, high);
     } /* if (element1 < high) */
     printf("After sorting, %d through %d is:\n",
         low, high);
     print_array(array, length);
} /* quicksort_helper */
```

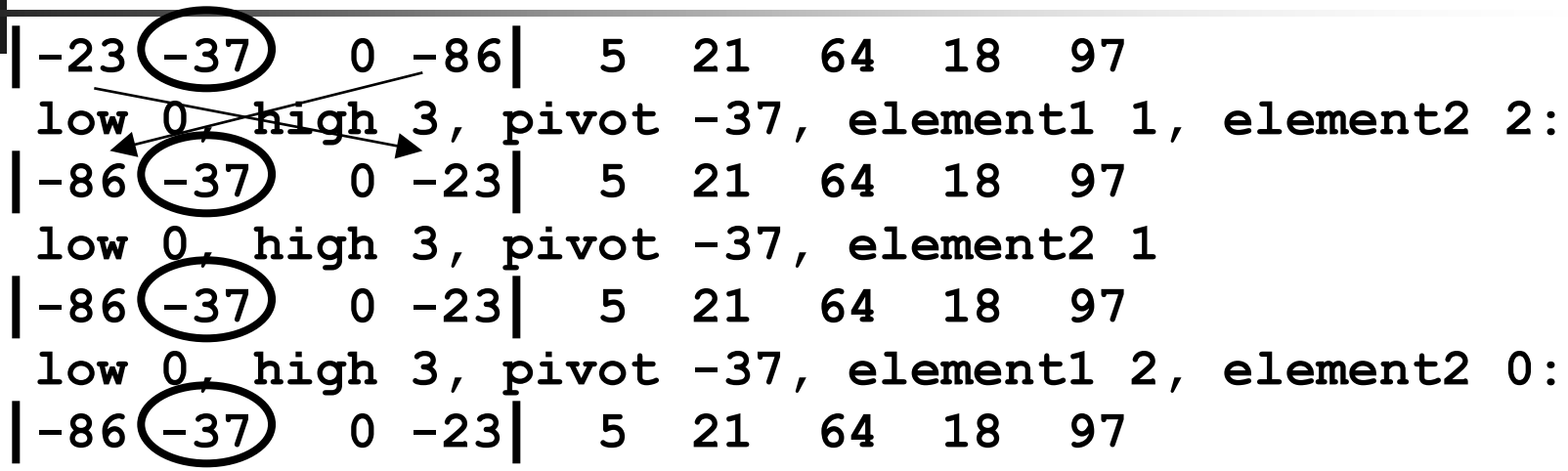
Quick Sort Run Start

```
I'm going to sort an array for you.
How many elements will the array have?
9
What are the 9 elements, unsorted?
-23  97  18  21   5 -86  64   0 -37
```

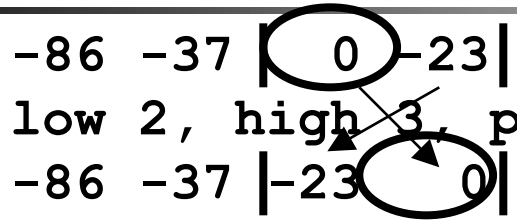
Quick Sort Run Call: low 0, high 8



Quick Sort Run Call: low 0, high 3



Quick Sort Run Call: low 2, high 3

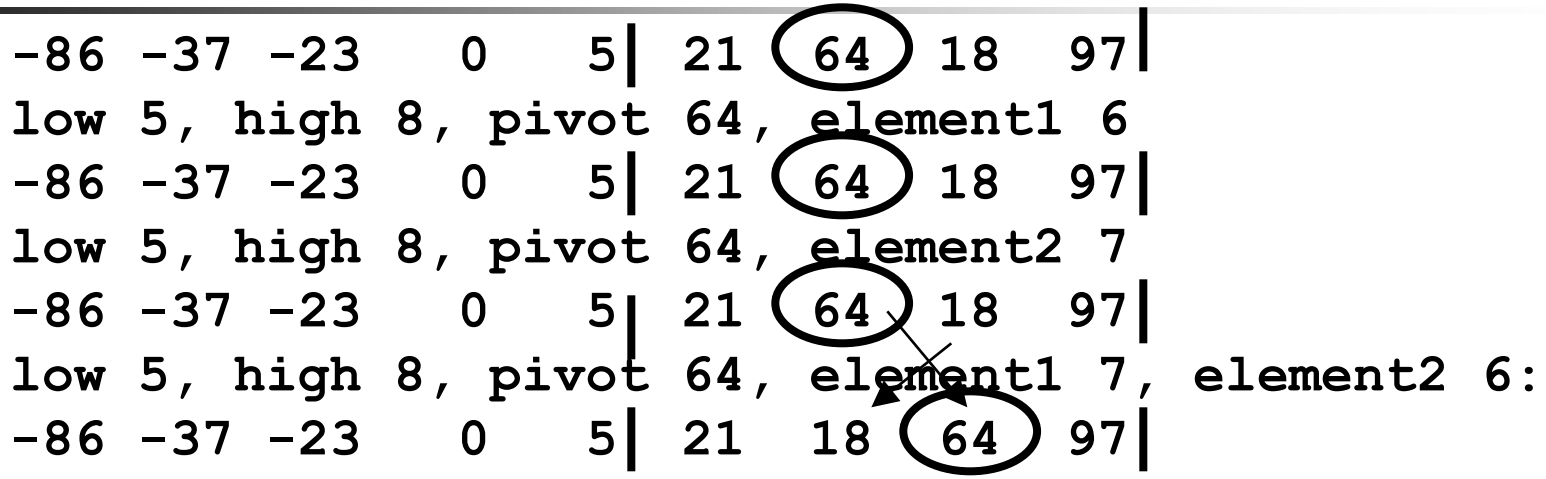



```
After sorting, 2 through 3 is:
-86 -37 -23   0  5  21  64  18  97
```

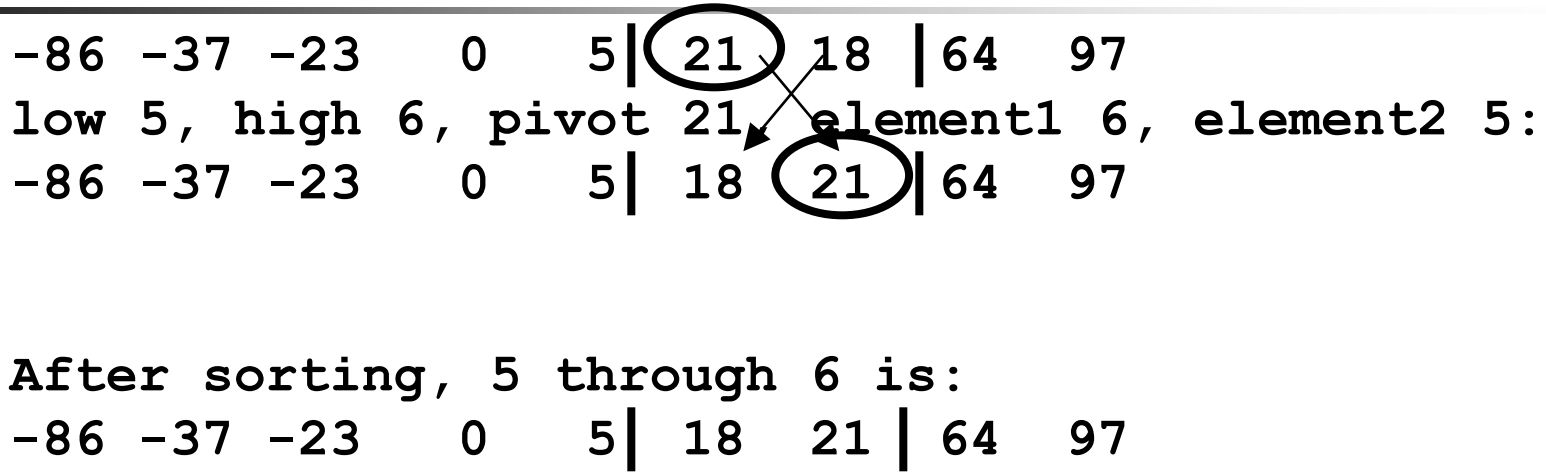
Quick Sort Run Call: low 0, high 3 End

```
 After sorting, 0 through 3 is:
|-86 -37 -23   0|  5  21  64  18  97
```

Quick Sort Run Call: low 5, high 8



Quick Sort Run Call: low 5, high 6



Quick Sort Run Call: low 7, high 8

```
-86 -37 -23   0   5  18  21| 64  97|
low 7, high 8, pivot 64, element2 7
-86 -37 -23   0   5  18  21| 64  97|
low 7, high 8, pivot 64, element1 8, element2 6:
-86 -37 -23   0   5  18  21| 64  97|

After sorting, 7 through 8 is:
-86 -37 -23   0   5  18  21| 64  97|
```

Quick Sort Run Call: low 5, high 8 End

```
After sorting, 5 through 8 is:
-86 -37 -23   0   5 |18  21  64  97|
```

Quick Sort Run Call: low 0, high 8 End

```
 After sorting, 0 through 8 is:
|-86 -37 -23   0   5  18  21  64  97|
```

Quick Sort Run End

```
The 9 elements, sorted, are:
-86 -37 -23   0   5  18  21  64  97
```

Time Complexity of Quick Sort #1

What is the time complexity of quick sort?

Suppose the array has length *n*.

Well, each call to `quicksort_helper` cuts the array in half.

But, each call to `quicksort_helper` can be accompanied by another call that covers the other half of the current subarray.

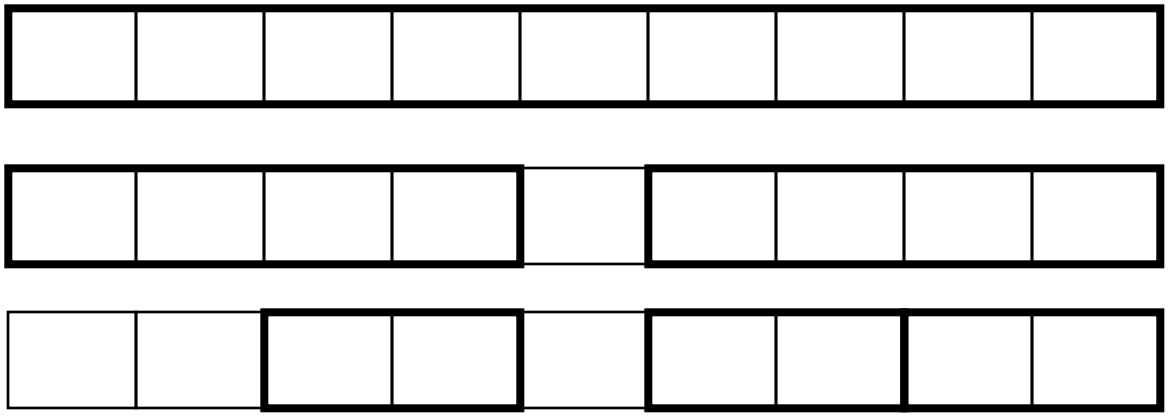
So, at each level of calls, most of the array gets visited.

So all we need to know is, how many levels of calls are there?

Time Complexity of Quick Sort #2

At each level of calls, the entire array gets visited.

All we need to know is, how many levels of calls are there?



It turns out that, for an array of length n, there are $\mathbf{O}(\log n)$ levels.

So the total time complexity is $\mathbf{O}(n \log n)$ – it's **quick**!