Sorting

2501ICT/7421ICTNathan

René Hexel

School of Information and Communication Technology
Griffith University

Semester 1, 2012

Outline

Sort Algorithms

- Many Sort Algorithms Exist
- Simple, but inefficient
- Complex, but efficient
- Two Case Studies
- Bubble Sort
  - “bubble the largest items to the end of the array”
- Mergesort
  - Divide-and-Conquer strategy

Bubble Sort

- Simple to Implement
- Slow, Inefficient Algorithm

Strategy

- Compare pairs of data items
- If pairs are out of order, swap them
- Repeat from beginning to end of array

Bubble Sort in Objective-C

Example

```
#import <Foundation/Foundation.h>


#define SWAP(arr, x, y) do {                                                    \
        id oldX = [arr objectAtIndex:  (x)];                                     \
        [arr replaceObjectAtIndex:  (x) withObject:  [arr objectAtIndex:  (y)]];   \
        [arr replaceObjectAtIndex:  (y) withObject:  oldX];                      \
} while (0)


void bubbleSort(NSMutableArray *array)
{
        int n = [array count];                          // n elements in array

        for (int i = 0; i < n-1; i++)                    // n-1 bubbling rounds
                for (int j = 0; j < n-i-1; j++)          // for each element
                        if ([[array objectAtIndex:  j] compare:
                                [array objectAtIndex:  j+1]] == NSOrderedDescending)
                                SWAP(array, j, j+1);    // bubble if out of order
}
```

Using `bubbleSort()`

Example (prints: `Array:  1 2 3 `)

```
/*
* A main() function for testing bubbleSort() from the previous slide
*/
int main(int argc, char *argv[])
{
    @autoreleasepool
    {
        NSMutableArray *array = [[NSArray arrayWithObjects:  @"3", @"1", @"2", nil]
                mutableCopy];                    // create a mutable array

        bubbleSort(array);                       // sort the array

        printf("Array:  ");
        NSString *element;
        NSEnumerator *iterator = [array objectEnumerator];
        while ((element = [iterator nextObject]) != nil)
                printf("%s ", [element UTF8String]);
        printf("\n");

        [array release];                         // array needs to be released!
    }

    return EXIT_SUCCESS;
}
```

Bubble Sort in C++

Example

```
#include <vector>

using namespace std;

#define SWAP(T, arr, x, y) do {                                    \
        T oldX = arr[x];                                           \
        arr[x] = arr[y];                                           \
        arr[y] = oldX;                                             \
} while (0)


template <class T> void bubbleSort(vector<T> &array)
{
        int n = array.size();                              // n elements in array

        for (int i = 0; i < n-1; i++)                       // number of bubble rounds
                for (int j = 0; j < n-i-1; j++)             // bubble each element
                        if (array[j] > array[j+1])          // if out of order
                                SWAP(T, array, j, j+1); // swap with neighbour
}
```

Using `bubbleSort()`

Example (prints: `Array:  1 2 3 `)

```
/*
* A main() function for testing bubbleSort() from the previous slide
*/
int main(int argc, char *argv[])
{
        vector<int> array(3);                    // create a mutable array
        array[0] = 3;                           // and fill with values
        array[1] = 1;
        array[2] = 2;

        bubbleSort(array);                      // sort the array

        cout << "Array:  ";
        vector<int>::iterator enumerator = array.begin();
        while (enumerator != array.end())
        {
                int element = *enumerator;
                cout << element << " ";
                enumerator++;
        }
        cout << endl;

        return EXIT_SUCCESS;
}
```

Bubble Sort Complexity Analysis

- Nesting Loop containing the privileged Instruction
- Total Number of Comparisons
  - $\rightarrow \frac{1}{2}n^2 - \frac{1}{2}n$

$\Rightarrow$ Complexity: $O(n^2)$

- Worst-Case behaviour, but also
- Best-Case behaviour!
  - $\rightarrow$ possible to add an `if`, but this only affects the best-case behaviour, not the average or worst case!

Mergesort

- Merge sort is more complex to implement than BubbleSort
- Much more efficient

Divide-And-Conquer Strategy

- Compute middle position
- Recursively sort left and right sub-arrays
  - $\Rightarrow$ becomes trivial for sub-arrays of size zero or one
- Merge sub-arrays back into single sorted array

Merge Sort in Objective-C (1)

Example (Divide and Conquer)

```
void mergeSort(NSMutableArray *array)
{
        NSMutableArray *buffer = [array mutableCopy];    // helper buffer
        int n = [array count];                           // number of items in array

        divideAndMerge(array, buffer, 0, n-1);           // sort everything
        [buffer release];                                // release the helper buffer
}

void divideAndMerge(NSMutableArray *a, NSMutableArray *b, int lo, int hi)
{
        int mi = (lo + hi) / 2;                  // calculate middle point

        if (lo >= hi) return;                     // size is 1 or less --> done

        divideAndMerge(a, b, lo,   mi);          // sort lower half
        divideAndMerge(a, b, mi+1, hi);          // sort upper half
        merge(a, b, lo, mi, hi);                 // merge back sorted halves
}
```

Merge Sort in Objective-C (2)

Example (merge two sorted sub-arrays)

```
void merge(NSMutableArray *a, NSMutableArray *b, int lo, int mi, int hi)
{
    int i, i1 = lo, i2 = mi + 1;          // sub-array indexes

    for (i = lo; i <= hi; i++)            // for each element in destination b
    {
        if (i1 > mi)                      // if i1 is out of bounds, copy from i2
            [b replaceObjectAtIndex:  i withObject:  [a objectAtIndex:  i2++]];
        else if (i2 > hi)                 // if i2 is out of bounds, copy from i1
            [b replaceObjectAtIndex:  i withObject:  [a objectAtIndex:  i1++]];
        else if ([[a objectAtIndex:  i1]   compare:  [a objectAtIndex:  i2]]
                == NSOrderedAscending)    // if (a[i1] < a[i2])  copy from i1
            [b replaceObjectAtIndex:  i withObject:  [a objectAtIndex:  i1++]];
        else                              // if (a[i1] >= a[i2]) copy from i2
            [b replaceObjectAtIndex:  i withObject:  [a objectAtIndex:  i2++]];
    }

    // copy back b into a --> same as [a setArray:  b];
    for (i = lo; i <= hi; i++)
            [a replaceObjectAtIndex:  i withObject:  [b objectAtIndex:  i]];
}
```

Using `mergeSort()`

Example (prints: `Array:  A B C `)

```
/*
* A main() function for testing mergeSort() from the previous slides
*/
int main(int argc, char *argv[])
{
    @autoreleasepool
    {
        NSMutableArray *array = [[NSArray arrayWithObjects:  @"C", @"A", @"B", nil]
                mutableCopy];                   // create a mutable array

        mergeSort(array);                       // sort the array

        printf("Array:  ");
        NSString *element;
        NSEnumerator *iterator = [array objectEnumerator];
        while ((element = [iterator nextObject]) != nil)
                printf("%s ", [element UTF8String]);
        printf("\n");

        [array release];                        // array needs to be released!
    }

    return EXIT_SUCCESS;
}
```

Merge Sort in C++ (1)

Example (Divide and Conquer)

```
template <class T> void mergeSort(vector<T> &array)
{
        unsigned n = array.size();                      // n elements in array
        vector<T> buffer(n);                             // helper buffer

        divideAndMerge<T>(array, buffer, 0, n-1);        // sort everything
}

template <class T>
void divideAndMerge<T>(vector<T> &a, vector<T> &b, int lo, int hi)
{
        int mi = (lo + hi) / 2;                 // calculate middle point

        if (lo >= hi) return;                    // size is 1 or less --> done

        divideAndMerge<T>(a, b, lo,   mi);       // sort lower half
        divideAndMerge<T>(a, b, mi+1, hi);       // sort upper half
        merge<T>(a, b, lo, mi, hi);              // merge back sorted halves
}
```

Merge Sort in C++ (2)

Example (merge two sorted sub-arrays)

```
template <class T> void merge(vector<T> &a, vector<T> &b, int lo, int mi, int hi)
{
  int i, i1 = lo, i2 = mi + 1;                    // subarray indexes

  for (i = lo; i <= hi; i++)                       // for elements in destination b
  {
    if (i1 > mi)                 b[i] = a[i2++]; // i1 is out of bounds
    else if (i2 > hi)            b[i] = a[i1++]; // i2 is out of bounds
    else if (a[i1] < a[i2])      b[i] = a[i1++]; // choose smaller element at i1
    else                        b[i] = a[i2++]; // choose smaller element at i2
  }

  for (i = lo; i <= hi; i++)     a[i] = b[i];     // re-create a from b
}
```

Using `mergeSort()`

Example (prints: `Array:  T U V `)

```
/*
* A main() function for testing mergeSort() from the previous slide
*/
int main(int argc, char *argv[])
{
        vector<string> array(3);                  // create a mutable array
        array[0] = "U";                          // and fill with values
        array[1] = "T";
        array[2] = "V";

        mergeSort(array);                         // sort the array

        cout << "Array:  ";
        vector<string>::iterator enumerator = array.begin();
        while (enumerator != array.end())
        {
                string element = *enumerator;
                cout << element << " ";
                enumerator++;
        }
        cout << endl;

        return EXIT_SUCCESS;
}
```

Mergesort Complexity Analysis

- Two, Non-Nested for Loops
  - $O(n)$
- Subarrays are halved each stage
  - $O(\log n)$ stages
- Total Complexity:

  $\Rightarrow$ $O(n) \cdot O(\log n) = O(n \log n)$
- Best Possible Complexity