

Sorting “Recap”

What is a natural way to sort?

Sorting:

- stacks of exams?
 - want it easier to enter in a grade book and return in class?
- a hand of playing cards?
 - want to be able to plan your strategy?
- a deck of playing cards?
 - make sure no cards are missing?
- a case of collector cards?
 - so you can make full sets?
- others?

Would any/all of these work as computer algorithms?

What if our data is held in arrays?

What is the input?

What do we need as input to a sorting algorithm?

- A list of values is the obvious thing we need. For our initial discussions, this will be an array-based list.
- Is there anything else?

Some Sorting Algorithms

Some examples you've probably already seen:

- BubbleSort
- SelectionSort
- InsertionSort

With what similar philosophy do all these algorithms approach the problem of sorting?

InsertionSort

Input: list of values

Output: ordered list of values

Algorithm:

- Start with a one-element sorted list.
- Take “next” value and insert it in the correct place of the already-sorted list.
- Repeat above until all values have been inserted.

InsertionSort Pseudocode

InsertionSort(L) {

/* start with L[1] as a one-element list that
is already sorted */

for pos = 2 to L.length {

val = L[pos];

/* insert val in the correct place in the
already ordered sublist, sliding elements
over as you search */

iter = pos-1;

while (iter <> 0) and (L[iter] > val) {

L[iter+1] = L[iter];

iter--;

}

L[iter+1] = val;

} //endfor

}

Analysis of InsertionSort

It is not recursive, so you can use summations to represent the for and while loops...

More sorting algorithms...

Some other examples you've probably already seen:

- MergeSort
- QuickSort

With what similar philosophy do both of these algorithms approach the problem of sorting?

MergeSort

An example of a Divide & Conquer algorithm.

- Split the list in half
- Sort each half
- Merge them back together

Common MergeSort Pseudocode

```
MergeSort (L, start, size) {  
    if (size>1) {  
        middle = Floor(size/2);  
        MergeSort(L,start,middle);  
        MergeSort(L,start+middle,size-middle);  
        Merge(L,start,middle,  
              start+middle,size-middle);  
    }  
}
```

This algorithm re-uses the array holding the original list as it works.

Merge(L1,L2)

Since this sorting algorithm requires us to merge two array-based lists (stored in the same actual array in memory) we should discuss that as well.

“Thought Question” – Is it possible to perform an efficient merge of two logical sub-lists without using a large amount of temporary space of some sort in the array-based MergeSort?

```

Merge (Lst, left, left_size, right, right_size) {
new Array[left_size] L;
new Array[right_size] R;
for i = 1 to left_size L[i]=Lst[left+i];
for i = 1 to right_size R[i]=Lst[right+i];

posL=1;
posR=1;
posLst=left;
while (posL<=left_size)&&(posR<=right_size)
    if L[posL]<R[posR]
        Lst[posLst++]=L[posL++];
    else
        Lst[posLst++]=R[posR++];

if (posR<=right_size)
    for i=posR to right_size Lst[posLst++]=R[i];
else
    for i=posL to left_size Lst[posLst++]=L[i];
}

```

**What is the run-time of Merge in terms of
data comparisons?**

Analysis of MergeSort

```
if (size>1) {  
    middle = Floor(size/2);  
    MergeSort(L,start,middle);  
    MergeSort(L,start+middle,size-middle);  
    Merge(L,start,middle,  
          start+middle,size-middle);  
}
```

Looking at comparisons:

$$T(1) = ???$$

$$T(n) = ???$$

NOTE: This is recursive, so our time on input of size n will be a recurrence relation!

InsertionSort -vs- MergeSort

Looking at comparison-counting only:

Who has the better best case?

Worse case?

Average case?

Are there other factors to consider?

Do these other factors matter
asymptotically when comparing two
algorithms?

Even more algorithms..

There are many more sorting algorithms out there...

- RadixSort
- BucketSort
- SpaghettiSort
- LUPsort

There are also algorithms designed specifically for multi-processor systems.

Does MergeSort lend itself to some parallelism easily? What issues might arise?