

CIS 213: Foundations of Computer Science—Lab 6

Outline of activities:

- Discuss the MergeSort and its analysis
- Analyze MergeSort

Getting Started

Create a lab6 directory in your cis213 directory. Goto your lab6 directory and Copy the file from the lab directory, `~/hardnett/pub/cis213/lab6` into your lab6 directory. The file is:

1. mergesort-analysis.cc

First recap mergesort concept with an example:

L	N	K	A	B	G	H	F	P	D	C	E	J	I	M	O
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Show how the list is subdivided. See how the sublist are merged?

Compile to `mergesort-analysis` and run it for various values of n . Keep n fairly small (less than 100). On each execution, answer 'y' to the details question a few times to try to understand the *depth* of the recursion, and when the results are being merged.

```
host% mergesort-analysis 10
```

Analyzing Mergesort

First, an informal analysis:

- How much work is it to split a list in half?
- How many times do we do that?
- How much work is it to merge two lists?
- How much work gets done on each level?
- How many levels?

Now, a precise analysis of merge:

```
LIST merge(LIST A, LIST B) {
1   if( A==NULL ) return B;
2   if( B==NULL ) return A;
3   if( A->elt < B->elt ) {
4       A->next = merge(A->next,B);
5       return A;
6   } else {
7       B->next = merge(A,B->next);
8       return B;
9   }
}
```

- base:

$$T(0) =$$

- induction:

$$T(n) =$$

Analysis of split:

```

void split( node* & list, node* & odds, node* & evens) {
1  if( NULL==list ) {
2    odds = NULL;
3    evens = NULL;
4    return;
  }
5  if( NULL==list->next ) {
6    odds = list;
7    evens = NULL;
8    list = NULL;
9    return;
  }
10 odds = list;          // first element is odd
11 evens = list->next;   // second element is even
12 node* rest_odds;     // the odd elements
13 node* rest_evens;    // the even elements
14 split( list->next->next, rest_odds, rest_evens );
15 odds->next = rest_odds;
16 evens->next = rest_evens;
  }

```

- basis:

$$T(0) =$$

- basis:

$$T(1) =$$

- induction:

$$T(n) =$$

Analysis of MergeSort:

```

node* MergeSort(node* list) {
1  if( NULL==list || NULL==list->next ) return list;
2  node* odds;
3  node* evens;
4  split(list,odds,evens);
5  return merge(MergeSort(odds),MergeSort(evens));
  }

```

- basis:

$$T(1) = O(1)$$

- induction:

$$T(n) = 2t(n/2) + O(n)$$

We can show that

$$T(n) = 2^i T(n/2^i) + ibn$$

We hit the base case when $2^i = n$ or when $i = \log_2 n$. Hence

$$T(n) = 2^{\log_2 n} T(n/2^{\log_2 n}) + (\log_2 n)bn$$

But by definition, that's just:

$$T(n) = nT(1) + bn \log_2 n$$

Since the base case is $T(1) = a$, we get:

$$T(n) = an + bn \log_2 n$$

So,

$$T(n) = O(n \log n)$$

To do: Insert code to count the total number of comparisons between two list elements. (This constitutes our “inner loop.”) How many comparisons does it do? How well does it approximate $n \log n$? Compute and print that out for comparison.

You can calculate $n \log_2 n$ in your program as follows:

```
#include <math.h>
```

```
float log2n = log(n)/log(2);
```

The `log` function computes the natural logarithm of the argument. Dividing by `log 2` converts to base 2 logarithms. (Make sure you understand that!).

Submit your lab when you are done. The submission will only be the program files:

```
host% turnin cis213 lab6
```

Be sure that you see that your `mergesort-analysis.cc` file was submitted.