Iteration method for recurrences

- Decompose recurrence into a series of terms, and derive the \( n^{th} \) expression from the previous ones.

**Example:** Binary Search (half-interval/logarithmic search)

```c
binary_search(A, lo, hi, x)
{
    if (A[lo] > A[hi])
        return false
    mid = (lo+hi)/2
    if (x = A[mid])
        return true
    if (x < A[mid])
        binary_search(A,lo,mid-1,x)
    if (x > A[mid])
        binary_search(A,mid+1,hi,x);
}
```

\[
A = \{1, 2, 3, 4, 5, 7, 9, 11\} \quad n=8
\]

\[x = 7 \quad lo = 1 \quad hi = 8\]

\[
\text{mid} = \frac{1 + 8}{2} = 4.5 \quad \text{integer mid} = 4
\]

\[A[\text{mid}] = A[4] = 4\]
\[ \text{mid} = \lfloor (5+8)/2 \rfloor = \lfloor 6.5 \rfloor = 6 \]
\[ A[\text{mid}] = A[6] = 7 \]
\[ A[\text{mid}] = A[6] = 7 \]

\[ X = 6 \]

From the binary search alg. above...

\[ T(n) = C + T(n/2) \]

Find explicit formula of \( T(n) \) using the iteration method.

\[ T(n) = C + T(n/2) \]
\[ = C + C + T(n/4) \]
\[ = C + C + C + T(n/8) \]
\[ = C + C + C + C + T(n/2^k) \]

\[ = kC + T(n/2^k) \]

Assuming \( n = 2^k \), then \( k = \log n \)

\[ T(n) = C \log n + T(1) \]

Thus \( T(n) = O(\log n) \)
Merge sort

- efficient sorting alg.
- input
  \( \rightarrow \) unsorted array \( E \)
  \( \rightarrow \) integers \( \text{first}, \text{last} \)
- output
  \( \rightarrow \) array \( E \) containing a permutation of the input such that
  \[ E[\text{first}] \leq E[\text{first}+1] \leq \ldots \]
  \[ E[\text{last}-1] \leq [\text{last}] \]

**Idea:**

1. Divide inserted list (array) into \( n \) sublists, each containing one element (a list of one element is considered sorted)
2. Repeatedly merge sublists to produce new sorted sublists until there is 1 sublist remaining (this is the sorted list)

Merge sort uses the "merge" func., built into C++ STL

```c++
merge_sort(E, first, last)
{
    if (first<last)
        mid=floor(first+last)/2;
        merge_sort(E,first,mid);
        merge_sort(E,mid+1,last);
        merge(E,first,mid,last);
}
```
Drop constant time c in favor of larger time n

\[ T(n) = n + 2 T(n/2) \]

Now use iteration to solve

\[ T(n) = n + 2 T(n/2) = n + 2(\frac{n}{2} + 2 T(n/4)) \]
\[ = n + n + 4 T(n/4) \]
\[ = n + n + 8 T(\frac{n}{8}) \]
\[ = 3 n + 2^3 T(n/2^3) \]
\[ = 4 n + 2^4 + T(n/2^4) \]
\[ = 8 n + 2^5 T(n/2^5) \]

Change of variables

IDEA: We transform the recurrence to one that we have seen before.

1. Solve the recurrence
   \[ T(2^m) = 2T(2^m) + \log_2 n \]
   Where \( m = \log_2 n \) and \( n = 2^m \)
   \[ T(2^m) = S(2^m) + m \]

2. Recurrence \( k = m \) and \( S(k) = T(2^k) \).

   \[ S(k) = 2 S(k/2) + k \log_2 k \]

   From the previous example, we have
   \[ S(k) = O(k \log k) \]

   Thus, \( T(n) = T(2^n) \)
   \[ = S(m) = O(m \log m) = O(\log n \log(\log n)) \]