

from before:

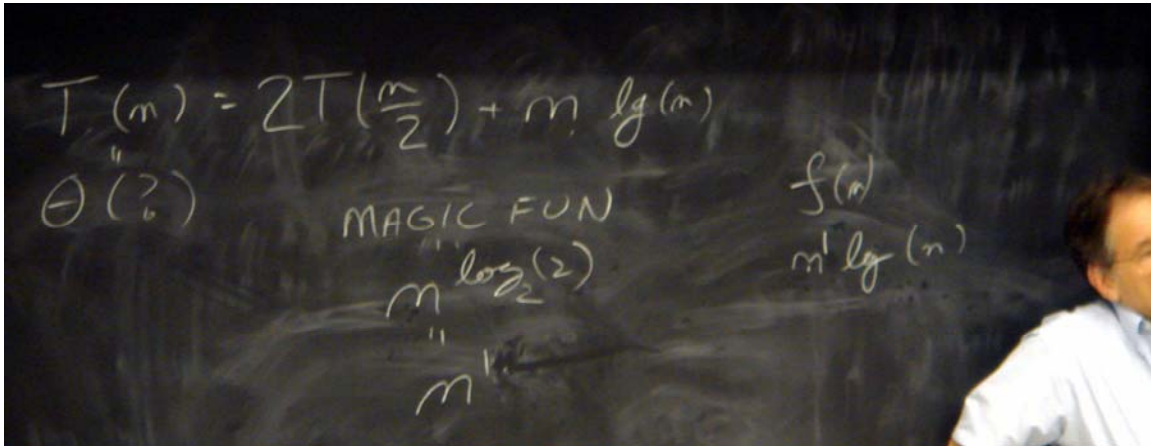
$$T_{MS}^{WC}(n) \leq c_1 + c_2 + 2T_{MS}^{WC}\left(\frac{n}{2}\right) + T_{MERGE}^{WC}(n).$$

The magic function  $(n^{\log_b a})$  has to be polynomially larger (or smaller) than  $f(n)$  for cases I (or III) to apply. That is, there has to be some  $\varepsilon > 0$  such that  $n^{\log_b a - \varepsilon}$  is larger (or than  $n^{\log_b a + \varepsilon}$  is smaller) than  $f(n)$ . Thus the Master Theorem does not apply to

$$T(n) = 2T\left(\frac{n}{2}\right) + n \lg n.$$

As a rule of thumb for this class, compare the powers of  $n$  in the magic function and  $f(n)$ . If they are different, the Master Theorem applies. If they are the same, the Master Theorem applies only if there are no log factors in  $f(n)$ . Otherwise, the Master Theorem does not apply.

Note: Master Theorem asks if there exists an  $\varepsilon > 0$ , not for all  $\varepsilon > 0$ . Also, word “any” is ambiguous: avoid using it.



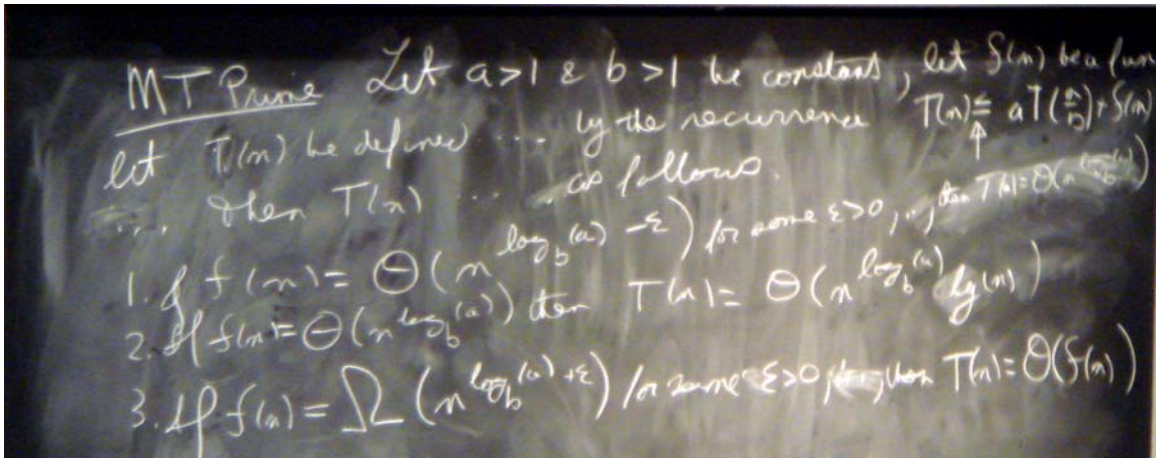
Master Theorem Prime:

Let  $a \geq 1, b > 1$  be constants, let  $f(n)$  be a function, let  $t(n)$  be defined on the non-negative integers by the recurrence  $T(n) \leq aT\left(\frac{n}{b}\right) + f(n)$ , where we interpret  $\frac{n}{b}$  to mean either

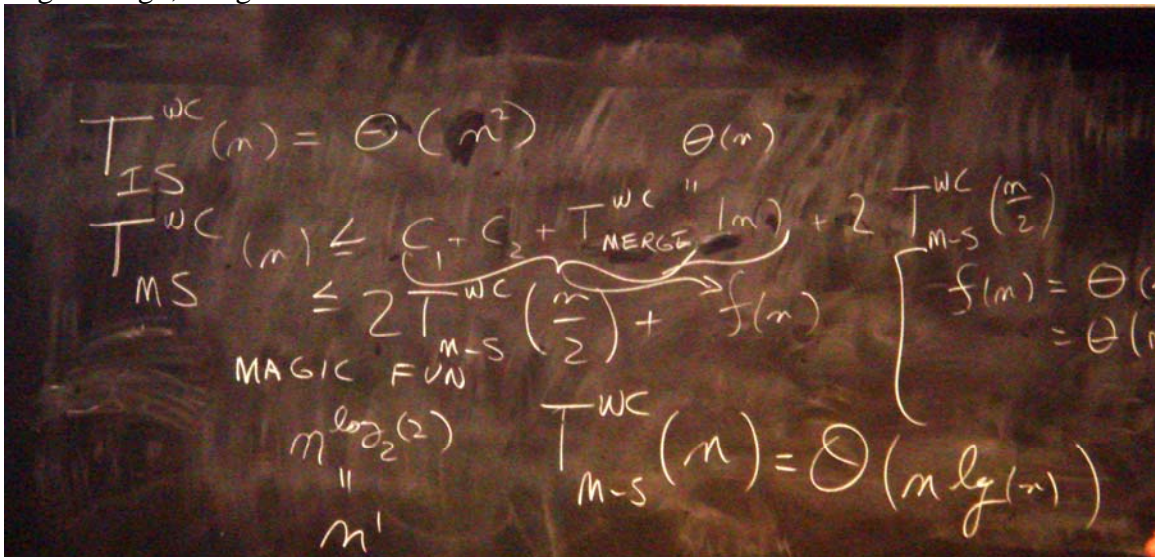
$\left\lfloor \frac{n}{b} \right\rfloor$  or  $\left\lceil \frac{n}{b} \right\rceil$ . Then  $T(n)$  can be bounded asymptotically as follows:

1. If  $f(n) = O(n^{\log_b a - \varepsilon})$  for some  $\varepsilon > 0$ , then  $T(n) = O(n^{\log_b a})$
2. If  $f(n) = \Theta(n^{\log_b a})$ , then  $T(n) = O(n^{\log_b a} \lg n)$

3. If  $f(n) = \Omega(n^{\log_b a + \epsilon})$  for some  $\epsilon > 0$ , then  $T(n) = O(f(n))$



Thus,  $T_{MS}^{WC}(n) = O(n \lg n)$ . So asymptotically, Merge Sort is better than Insertion Sort. That means that for all implementations of Merge Sort and Insertion Sort, for all inputs large enough, merge sort will be faster.



Let's examine the difference between Merge Sort and Insertion Sort. Suppose we used every operation ever made by a human made machine to sort numbers.



We estimate there have been no more than  $128 * 10^{25} \approx 2^{90}$  such operations. So an  $n^2$  algorithm could sort no more than  $2^{45}$  numbers, while an  $n \lg n$  algorithm could sort at least  $2^{83}$  numbers. That's a huge difference! So faster algorithms are incredibly important to solving large problems. At least as important as making computer chips faster.

### Divide and Conquer

Search Problem:

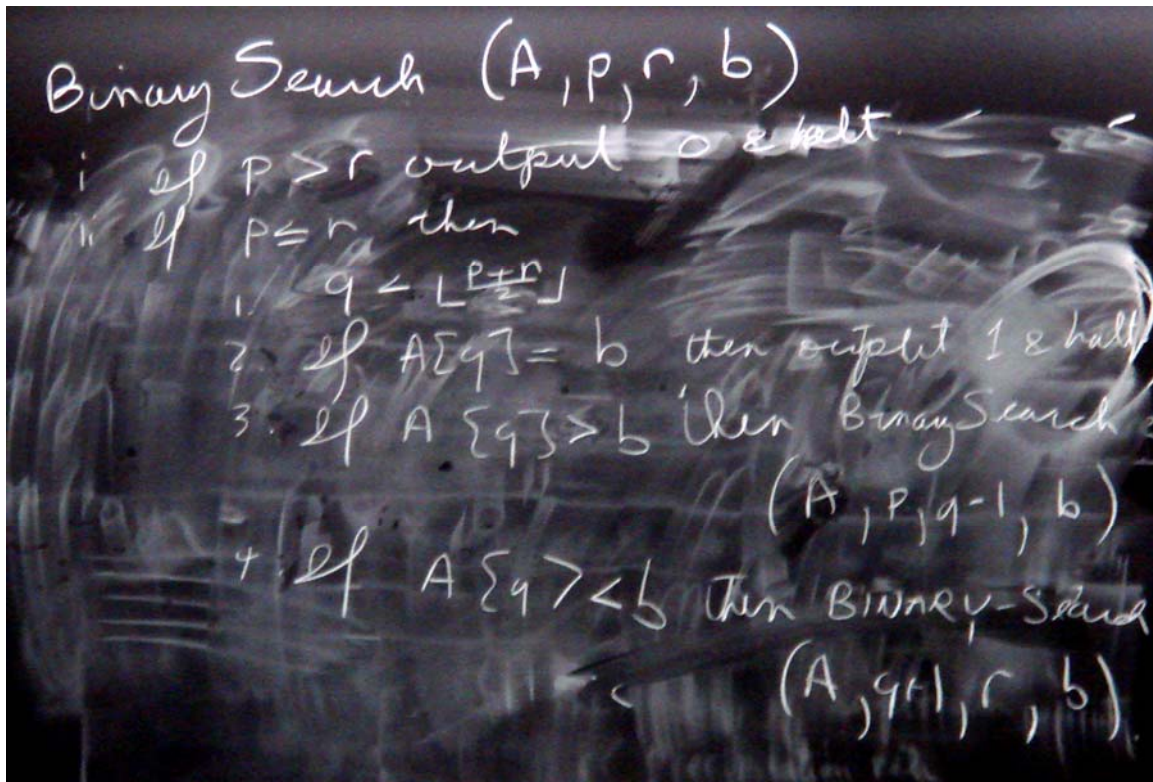
Input:  $T, a_1, a_2, \dots, a_n$  such that  $a_1 < a_2 < \dots < a_n$ .

Output: "yes" if  $\exists i[a_i = T]$ , and "no" otherwise.

Linear search: compare  $T$  to each number.

$$T_{LS}^{WC}(n) = \Theta(n)$$

Binary search: compare  $T$  to the middle number  $\left( a_{\lfloor \frac{n+1}{2} \rfloor} \right)$ , and recursively search on the correct half of the input.



Binary Search(A, p, r, b)

i. if  $p > r$  then output 0 and halt.

ii. if  $p \leq r$  then

1.  $q = \left\lfloor \frac{p+r}{2} \right\rfloor$

2. if  $A[q] = b$  then output 1 and halt

3. if  $A[q] > b$  then Binary Search(A, p, q-1, b)

4. if  $A[q] < b$  then Binary Search(A, q+1, r, b)

$$T_{BS}^{WC}(n) \leq T_{BS}^{WC}\left(\frac{n}{2}\right) + c = O(\lg n).$$

# Divide & Conquer

INPUT:  $a_1, a_2, \dots, a_n, T$  st  $a_1 < a_2 < a_3 \dots < a_n$

OUTPUT: "Yes" if  $(\exists i) [a_i = T]$   
 "No" otherwise

$$T_{BS}^{TWC}(n) = C + T_{BS}^{TWC}\left(\frac{n}{2}\right)$$

$$\Theta(\log_2(n))$$

Recurrence  
 $T(n) = T\left(\frac{n}{2}\right) + C$