

◦

CS 170, Fall 2022

Contents

1. Algorithms with numbers	3
2. Divide-and-conquer Algorithms	4
2.a. Multiplication	4
2.b. Recurrence Relations	4
2.c. Mergesort	4
2.d. Medians	5
2.e. Matrix Multiplication	5
2.f. Fast Fourier Transform	5

1. Algorithms with numbers

2. Divide-and-conquer Algorithms

2.a. Multiplication

Definition 2.1 (Integer Multiplication)

A divide-and-conquer algorithm for integer multiplication is defined as follows:

```

1 function mul(x(0b[1...k]), y(0b[1...h]))
2 %Input: Positive integers x, y in binary
3 %Output: x times y
4
5 n = max(size of x, size of y)
6 if n == 1: return x * y
7
8 x_L, x_R = x(0b[1...[n/2]]), x(0b[[n/2]...n])
9 y_L, y_R = y(0b[1...[n/2]]), y(0b[[n/2]...n])
10
11 P1 = mul(x_L, y_L)
12 P2 = mul(x_R, y_R)
13 P3 = mul(x_L + x_R, y_L + y_R)
14 return P1 * 2^n + (P3 - P1 - P2) * 2^{n/2} + P2

```

Where $0b[1...k]$ denotes the binary string representing a number.

Each call to *mul* has three recursive calls, inputs of which are half the size of the original inputs, and the base cases (*x times y*) take constant time. Therefore we conclude that the time taken by this algorithm is

$$T(n) = 3T(n/2) + O(n)$$

If apply the Master Algorithm in Chap 2.b, has the time complexity

$$T(n) \in O(n^{\log_2 3}) \approx O(n^{1.585})$$

2.b. Recurrence Relations

Theorem 2.2 (Master Algorithm)

If $T(n) = aT(n/b) + cn^k$ and $T(1) = c$ for some constants a, b, c and k , then

$$T(n) \in \begin{cases} O(n^k) & \text{if } a < b^k \\ O(n^k \log n) & \text{if } a = b^k \\ O(n^{\log_b a}) & \text{if } a > b^k \end{cases}$$

2.c. Mergesort

Definition 2.3 (Mergesort)

The Mergesort algorithm is defined as follows:

```

1 function mergesort(a[1...n])
2   %Input: An array of numbers a[1...n]
3   %Output: Sorted array a
4
5   if n>1:
6     return merge(mergesort(a[1...[n/2]]), mergesort(a[[n/2]+1...n]))
7   else:
8     return a
9
10 function merge(x[1...k], y[1...h])
11   %Input: Two arrays of numbers (x[1...k], y[1...h])
12   %Output: An array of numbers in x and y in ascending order
13
14   if k=0: return y
15   if l=0: return x
16   if x[1] <= y[1]:
17     return x[1] ◦ merge(x[2...k], y[1...h])
18   else:
19     return y[1] ◦ merge(x[1...k], y[2...h])

```

Where \circ denotes concatenation.

The *merge* function above does a constant amount of work (concatenating two arrays) per recursive call, for a total running time of $O(k + l)$. Thus the calls to *merge* in *mergesort* are linear, we conclude that the overall time taken by *mergesort* is

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

Recall the Master Algorithm, here $a = b^k$, and therefore

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

2.d. Medians**2.e. Matrix Multiplication****Definition 2.4****2.f. Fast Fourier Transform**