A Theorist's Toolkit **(CMU 18-859T, Fall 2013)**

Lecture 5: Computational Models

September 23

Lecturer: Ryan O'Donnell *Scribe: Guru Guruganesh*

1 Motivation

Consider a well known problem such as sorting: given n integers output them in nondecreasing order. How fast will the best algorithm run? A common bound that is thrown about is that this process takes $\Theta(n \log n)$ time. However, algorithms such as radix sort or counting sort which claim to have running time of $O(n)$ time. The resolution to this apparent conflict is that sorting takes $\Theta(n \log n)$ in the comparison model – a model where we are only allowed to compare two input integers. In fact, once the model is established it is easy to see this fact. The upper bound is established by well known algorithms such as merge sort. To establish the lower bound, we note that there are $n!$ different possible inputs. Each comparison will allow us 1 bit of information. Thus to distinguish all $n!$ inputs, we would need at-least $\log n! \approx n \ln n$ comparisons. It is important to note that the runtime depended on the model used.

We will look at a simpler problem which is to determine if a given binary string $x \in \{0,1\}^n$ is a palindrome (i.e is the same when reversed). Consider the following pseudo-code for the problem:

Algorithm 1 Palindrome Pseudo-Code

```
for i =1 to n do
  if x[i] != x[n-i+1] then
    return NO
return YES
```

It is not entirely obvious what this program's running time would be. Since the variable i iterates from 1 to n, it requires $\log n$ bits to store. It is not evident that incrementing this can be done in $O(1)$ time (a more sophisticated analysis does show that it takes $O(1)$ amortized time). Comparing the two locations in memory could also take more time depending on the type of memory used. However, we know that when we run this program on modern day computers, it runs very quickly.

For these reasons, it is important to choose the right computational model to analyze the algorithm. In this lecture, we will discuss the three primary models that have been studied in theoretical computer science:

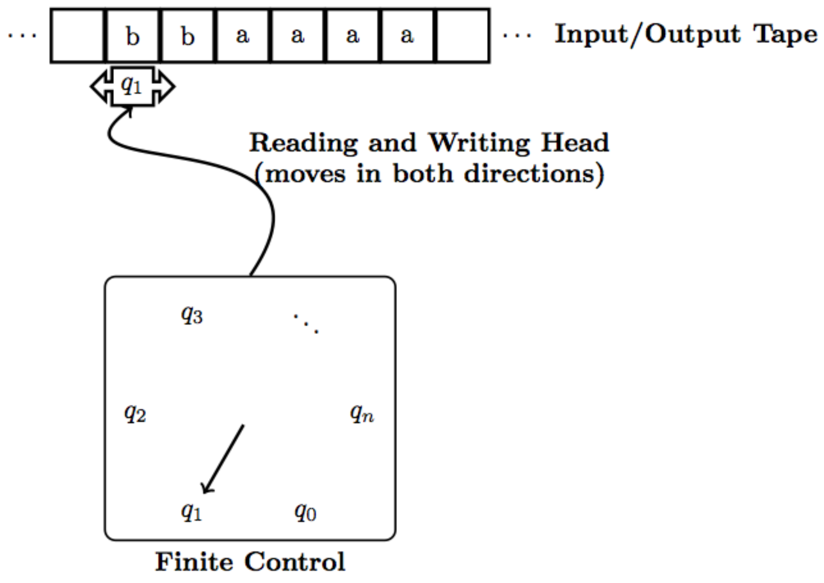
- Turing Model
- Circuits
- Word RAM Model

2 Turing Machines

2.1 Single Tape Turing Machines

We assume that the reader has had some exposure to the Turing Machine. Here is an informal definition. You have an infinite tape with symbols on it and a machine that has a finite control (i.e finite number of

Figure 1: A Turing Machine



states). This control has a tape head which can read/write the symbols and move one unit to the left/right per unit of time (see Figure 1 [1]). This model was first proposed by Alan Turing in 1936 as an abstract model about computation. He claimed that it abstracted a human sitting down performing calculations on sheet of paper.

Implementing Algorithm 1 on a single tape Turing Machine would be a little tricky. In particular, to check the condition inside the if statement, we would have to move the head from position i to position $n-i+1$. For each i this will take $O(n)$steps and hence the algorithm will take $O(n^2)$ steps in total. Can we do better?

Theorem 1. *[Hen65]The language palindrome cannot be recognized by any one-tape Turing Machine in time* $\Omega(n^2)$

This is very unsatisfactory as we know the code runs in a linear fashion when implemented on modern computers. Perhaps we could gain some advantage if we allowed the use of multiple tapes.

2.2 Multiple Tape Turing Machines

In a multi-tape turing machine, the head has access to several working tapes along with an input tape and an output tape.

Remark. In space complexity, the input tape is read only, the output tape is write only. The working tapes are read/write tapes. The space complexity measures the number of working tape squares are used. Hence it is possible to have sub-linear space usage.

Now we can solve the problem palindrome in linear timer. We can simply copy the input tape in reverse to the working tape in time $O(n)$. Then we could simply proceed and check if the input tape is the same as the working tape in time $O(n)$. Here is a pleasant theorem about the power of adding new tapes to Turing machines.

Theorem 2. *[HS66]If a k-tape Turing Machine can solve a problem in $O(T(n))$ then a 2-tape Turing Machine can solve the same problem in $O(T(n)\log n)$ and a 1-tape Turing Machine can solve the same problem in $O(T(n)^2)$.*

[1]The original figure was an example on from the site http://www.texample.net/tikz/examples/turing-machine-2/

Adding more tapes can give us at-most a quadratic sped-up. However, this is still very unsatisfactory as we use $\Omega(n)$ space to solve palindrome but in practice, we don't need extra space. Cobham captures our frustration in this theorem

Theorem 3. *[Cob66]Any $k-$tape (where $k \in \mathbb{N}$) Turing Machine that solves Pal simultaneously using time $T(n)$ and space $S(n)$ would satisfy $T(n) \cdot S(n) \in \Omega(n^2)$.*

2.3 Random Access Turing Machines

The reason we don't like Turing Machines is because they have a ridiculous memory model. We solve this by allowing random access to the tapes. In this model, each tape has an associated index, and the tape head is allowed an instruction where by it can move to the index of the tape which was specified by the current tape. Hence each spot in memory is randomly accessible by our Turing Machine. For a detailed definition, we refer the reader to [vM11]. This begins to capture what modern machines are capable of doing. It is known that palindrome can be solved in a random access turing machine in time $O(n)$ and in space $O(\log n)$. Lower bounds are more difficult to establish for Random Access Turing Machines. The theorem we discussed last class holds for Random Access Turing Machines.

Theorem 4. *[Wil07]For any $\epsilon > 0$, SAT cannot be solved simultaneously using time $T(n)$ and space $S(n)$ if $T(n) \cdot S(N) \in o\left(n^{2\cos(\pi/7)-\epsilon}\right)$.*

How long should $\max\{a_1, \ldots, a_n\}$ take? But input is $n \log n$ bits long. Doesn't this screw up our calculations? We will return to this later in Section 4.

3 Circuits

For the rest of this section we will assume that we are given a function $f : \{0,1\}^n \to \{0,1\}^m$

Definition 5. A basis is a set of functions which we will denote by$\mathcal{B}$. e.g $\mathcal{B} = \{\neg_1, \wedge_2, \vee_2\}$.

Definition 6. An circuit C is a sequence of t functions $g_1, g_2, \ldots g_n, g_{n+1} \ldots g_t$. The first n functions are the input variables (i.e $g_i(\alpha) = x_i$). Each subsequent g_j (also referred to as a gate) is an element of the basis $\mathcal{B}$ and is applied to the output of the previous g_i ($i < j$). Some $g_{i_1}, g_{i_2}, \ldots, g_{i_m}$describe outputs of the circuit.

The size of a circuit C is simply the number of non-input functions used (i.e Size $(C) = t - n$). The depth of a circuit C is the longest input-output path. The gates of a circuit form a directed acyclic graph (see Figure 2). Some of the most common basis are :

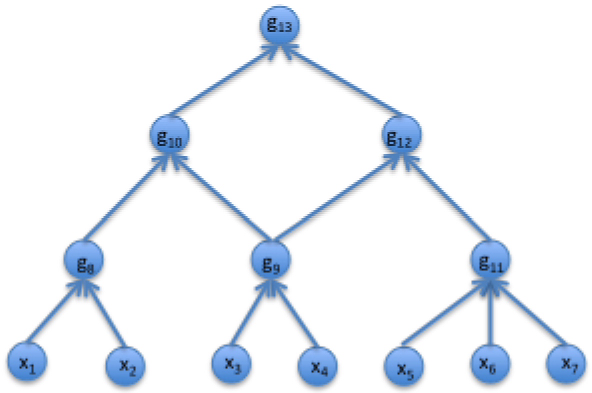
- $\mathcal{B}_2$ is the set of all 16 2-bit functions.
- U_2 (a.k.a Demorgan-circuits) consists of the set $\{\neg_1, \vee_2, \wedge_2\}$.
- $\{\neg_1, \wedge, \vee\}$ where $\wedge$ and $\vee$ have unbounded fan-in

For any function g_i, the fan-in is the number of inputs it requires and the fanout is the number of outputs it produces.

Definition 7. A formula is a function where all fan-outs have size 1.

Remark. In a formula, the directed acyclic graph produced is a tree.

Figure 2: A example circuit



When measuring circuit size, it is common to assume that there is some gate which produces 0 and a gate which produces 1 at no cost. In the U_2 basis, we also assume that $\neg$ gates are free. Unlike Turing Machines, one can be very precise about the size of circuits required to perform computation. For instance, the following theorem gives a very exact bound to compute languages that are a subset of a special type of second order logic.

Theorem 8. *[SM02] WS1S denotes a particular language in 2^{nd} order logic. To decide the truth of logical formulas of length at most $610 in$ WS1S requires a circuit containing at least 10^{125} gates.*[2]

It is well known that a circuit C with size s can be evaluated by a Turing Machine in time $O(s \log s)$. The equivalent converse was proved in the following theorem.

Theorem 9. *[PF79]If a TM evaluates length-n inputs in time $T(n)$, then there exists a circuit C with size $O(T(n) \cdot \log(T(n))$ which will evaluate the same function.*

Remark. The above theorem was first proven for oblivious Turing Machines but the result for circuit follows (see [Lip09]).

Theorem 10. *[Sha49]For any function $f : \{0,1\}^n \to \{0,1\}$ there exists a U_2-circuit of size $O(\frac{2^n}{n})$ which will evaluate f.*

Shannon used a counting argument to demonstrate this (achieving a bound of $O(2^n \cdot n)$ is easy). He also got a constant of 4 in the $O()$, which was later improved to $1 + o(1)$ in [Lup59]. Shannon's arguments also demonstrate that almost every function requires at-least $\Omega(\frac{2^n}{n})$ gates. Despite this result, we do not have an explicit family of functions that can compute $10n$! An explicit family is a set of functions $f : \{0,1\}^* \to \{0,1\}$ that belong to NP. 3-SAT is considered an explicit function. The best known bounds are

- [Blu83]For the basis B_2, there are functions computable in P (poly-time) which are known to require circuit size $3n - o(n)$.
- [IM02] For the basis U_2, there are functions computable in P (poly-time) which are known to require circuit size $5n - o(n)$.

[2]The original theorem was proven in the first author's thesis. The journal version provides the theorem in a historical context.

Even for the larger class of NP (non-deterministic polytime), the above bounds are the best known.

Circuits can be substantially different for each input size. This leads to some interesting applications such as being able to determine if a turing machine of size n will halt. Since there are only a finite number of turing machines that halt, there exists a list of all of them. Subsequently a circuit need only check if the given program matches any of the programs in the list. However, this does not mean that we can solve the Halting Problem. While we can solve the problem for any specific n, we cannot solve the problem for all n. In other words, there exists no turing machine that given n can produce as its output a circuit C_n, where each C_n can decide if a program of size n halts on a Turing Machine. In other words, no general method exists to generate an arbitrary circuit due to the fact that the halting problem is uncomputable. This gives rise to the definition of uniformity.

Definition 11. A circuit is uniform if there exists a function $f(n)$ which produces a circuit $(C_n)_{n \in N}$ and this function f can be computed by a turing machine in polynomial time.

Remark. Uniform circuits are equivalent to polynomial time Turing machines.

One important concept is that a circuit is more parallelizable is equivalent to saying it has low depth. One can think of each gate having its own processor, but has to wait for its inputs to arrive before it can finish its computation.

Definition 12. The set of circuits can be classified into several classes.
AC^0- poly(n) size circuits but $O(1)$ depth. and use unbounded fan-in Demorgan circuits.
NC^1- poly(n) size circuits but $O(\log n)$ depth using U_2 circuits
NC- poly(n) size U_2-circuits with polylog depths
$P/poly$- poly(n) size U_2-circuits with polynomial depth

It is known that $AC^0 \subseteq NC^1 \subseteq NC \subseteq P/poly$.
Let us consider some simple problems and which classes they belong to.

Problem Name	Description
Majority	Given n bits, output which bit occurs the most
Parity	Given n bits, output parity of all the bits
Binary Count	Given n bits, output in binary the number of 1s
Unary Count (sort)	Given n bits, output the number of 1s in unary.

Theorem 13. *[FSS84]Majority,Parity* $\notin AC^0$

All of the problems described in the table above belong in NC^1. There was a lot of excitement in the 80s that this approach could show that $P \neq NP$.

4 Word RAM Model

Recall the earlier question of computing the sum of n integers between $1..n$. This could take $O(n \log n)$ bits as we would need $\log n$ bits to compute the sum of two numbers (especially as n becomes large). Once again, this model does not really capture what we can do with our modern computers. To rectify this, we introduce the Word RAM Model which closely resembles the modern day computers. In this model, the memory is composed of words where each word is composed of w bits. Modern day machines uses words(a.k.a registers) with 64 bits.

It is not immediately clear that this helps since for any given w, we can generate arbitrarily large input sizes and make the model useless. However, it is generally assumed that $w \geq \log n$. If you are a bit uncomfortable with this idea, just pretend that all claims are prepended with the statement "Assume $w \geq \log n$". We also assume that in unit time we can do the following operations:

- $x + y \bmod 2^w$
- $x - y \bmod 2^w$

- bitwise AND,OR, XOR
- left/right cyclic shift by an arbitrary amount (lots of operations depend on this).
- multiplication?

Remark. It is a bit tricky to include multiplication in this list. One reason is that it is not in AC^0 while all the other operations are. People will include multiplication in their instruction set, if they really need it for their stated result but otherwise they won't.

Note that space is bounded by 2^w, because each jump instruction will be bounded by 2^w. It is a reasonable model to assume that $w \approx \Theta(\log n)$ and this special case has the name Transdichotomous Word RAM Model. Note that the sum of n numbers can be now computed in $O(n)$ time under this model.

4.1 Sorting

Let us consider sorting n numbers in range $0..n-1$. We could use counting sort which would allocate an array a of size n. Each entry $a[i]$ keeps count of how many i's have appeared in the input. At the end, we could run through the array to output a sorted list. Note that this algorithm (known as Counting sort) takes $O(n)$ time and $O(n)$space.
Suppose we are given n numbers in the range $0..2^w - 1$ and $w = 50\log n$. In this case, counting sort would use $O(n^{50})$ space and $O(n^{50})$ time as we would need to go through the array to output our result.

Remark. We might be able to reduce this down using some variant of counting sort and/or with the use of multiple arrays.

We could use $O(n)$ space and sort the numbers in $O(n \log n)$ time using merge sort (regardless of w). Perhaps we can do better using radix sort, which sorts the numbers by their least significant digits first. Here we would need $\frac{\log(n)}{w}$ many sorts on the last bits and thus would take time $O(\frac{nw}{\log n})$. Note that this sort becomes worse than merge sort as $w \to \infty$. Here is a list of some of the best bounds known for sorting under this model:

- [FW93]- $O(\frac{n \log n}{\log\log n})$ for all w (heavily uses multiplication).
- [Han04]$O(n \log w) \approx O(n \log\log n)$ for all non– values of w (did not use multiplication)
- [HT02]$O(n\sqrt{\log\log n})$ in the randomized case.
- [AHNR95]$O(n)$ time if $w \in \Omega(\log^{2+\epsilon} n)$

It remains an open question if one can do $O(n)$time for all w? For a more detailed discussion on the fastest sorting times, we refer the reader to [Wei12]. A lot is also known about "Dictionary" data structures, which beat the classic solution of balanced binary trees, see e.g [Dem12]. Other interesting properties have been shown in the Word RAM Model. For instance, single source shortest path can be performed in this model in $O(|E|)$ time for any w (see [Tho, Tho99]). For a general reference on Word RAM Model, we refer the reader to [Hag98].

5 Arithmetic in Various Models

Multiplication of two n-bit integers

We will list a few results for the minimum size circuit for the multiplication of two n bit numbers in the circuit model.

- $O(n^2)$ time for grade school method
- [KO63]like Strassen's method got it to $O(n^{\log_2 3})$ where $\log_2 3 \approx 1.5$

- [CA69, Too63]$O\left(f(\epsilon)\cdot n^{1+\epsilon}\right)$ $\forall\epsilon > 0$ where $\lim_{\epsilon\to 0} f(\epsilon) = \infty$
- [SS71]$O(n \log n \log\log n)$
- [Für09] $O(n \log n 2^{\log^* n})$ where $\log^*(n)$ is defined to be the number of logarithms needed before n gets down to 2.

If we assume that $w \approx \log n$ then we can perform multiplication in $O(n)$ time in the Word RAM Model (as shown in section 4.3.3.C of [Knu97]).

Fundamental Arithmetic

Since multiplication is just one operation, it is natural to ask how well we can perform other fundamental numerical operations. An excellent reference on how to compute these operations on various computational models is [BZ10]. We will survey some of their results here. Let us denote by $M(n)$ the time it takes to multiply two n-bit numbers together.

Remark. For non-integer arithmetic, we basically assume real numbers are stored/approximated by rational numbers as $m2^{\epsilon}$ where m and ϵ are (possibly negative) integers. One typically endeavors to get n bits of precision in m.

Dividing one number by another and getting n bits of precision takes $O(M(n))$ time and it is achieved by using Newton's Method. All the remaining operations we will discuss take $O(M(n)\log(M(n))$ time. We will discuss how they are achieved in the following table:

Operation	Technique
Square Root/ k^{th}root	Newton's Method
[Extended] Greatest Common Divisor on Integers	Does **not** use Euclidean Algorithm (which is quadratic)
Base Conversion	–
ln	Arithmetic-geometric mean iteration
exp / sin / cos / any other elementary function	Through computability of ln and Newton's Method
π, e	Through computability of ln and Newton's Method

References

[AHNR95] Arne Andersson, Torben Hagerup, Stefan Nilsson, and Rajeev Raman. Sorting in linear time? In *Proceedings of the twenty-seventh annual ACM symposium on Theory of computing*, pages 427–436. ACM, 1995.

[Blu83] Norbert Blum. A boolean function requiring $3n$ network size. *Theoretical Computer Science*, 28(3):337–345, 1983.

[BZ10] Richard P Brent and Paul Zimmermann. *Modern computer arithmetic*. Number 18. Cambridge University Press, 2010.

[CA69] Stephen A Cook and Stål O Aanderaa. On the minimum computation time of functions. *Transactions of the American Mathematical Society*, 142:291–314, 1969.

[Cob66] Alan Cobham. The recognition problem for the set of perfect squares. In *Switching and Automata Theory, 1966., IEEE Conference Record of Seventh Annual Symposium on*, pages 78–87. IEEE, 1966.

[Dem12] Erik Demaine. Lecture notes 12 for MIT Course 6.851 Advanced Algorithms. http://courses.csail.mit.edu/6.851/spring07/scribe/lec12.pdf, 2012.

[FSS84] Merrick Furst, James B. Saxe, and Michael Sipser. Parity, circuits, and the polynomial-time hierarchy. *Math. Systems Theory*, 17(1):13–27, 1984.

[Für09] Martin Fürer. Faster integer multiplication. *SIAM J. Comput.*, 39(3):979–1005, 2009.

[FW93] Michael L Fredman and Dan E Willard. Surpassing the information theoretic bound with fusion trees. *Journal of computer and system sciences*, 47(3):424–436, 1993.

[Hag98] Torben Hagerup. Sorting and searching on the word ram. In *STACS 98*, pages 366–398. Springer, 1998.

[Han04] Yijie Han. Deterministic sorting in o(nloglogn) time and linear space. *Journal of Algorithms*, 50(1):96 – 105, 2004.

[Hen65] Fred C Hennie. One-tape, off-line turing machine computations. *Information and Control*, 8(6):553–578, 1965.

[HS66] Fred C Hennie and Richard Edwin Stearns. Two-tape simulation of multitape turing machines. *Journal of the ACM (JACM)*, 13(4):533–546, 1966.

[HT02] Yijie Han and Mikkel Thorup. Integer sorting in $O(n\sqrt{\log\log(n)})$ expected time and linear space. In *Foundations of Computer Science, 2002. Proceedings. The 43rd Annual IEEE Symposium on*, pages 135–144. IEEE, 2002.

[IM02] Kazuo Iwama and Hiroki Morizumi. An explicit lower bound of 5n- o (n) for boolean circuits. In *Mathematical foundations of computer science 2002*, pages 353–364. Springer, 2002.

[Knu97] Donald E. Knuth. *The art or computer programming, volume 2 (3rd ed.): seminumerical algorithms.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1997.

[KO63] Anatolii Karatsuba and Yu Ofman. Multiplication of multidigit numbers on automata. In *Soviet physics doklady*, volume 7, page 595, 1963.

[Lip09] Richard J. Lipton. Oblivious Turing Machines and a "Crock". http://rjlipton.wordpress.com/2009/07/28/oblivious-turing-machines-and-a-crock/, July 2009.

[Lup59] Oleg B Lupanov. A method of circuit synthesis. *Izvestiya VUZ, Radiofizika*, 1(1):120–140, 1959.

[PF79] Nicholas Pippenger and Michael J. Fischer. Relations among complexity measures. *J. Assoc. Comput. Mach.*, 26(2):361–381, 1979.

[Sha49] Claude E. Shannon. The synthesis of two-terminal switching circuits. *Bell System Tech. J.*, 28:59–98, 1949.

[SM02] Larry Stockmeyer and Albert R. Meyer. Cosmological lower bound on the circuit complexity of a small problem in logic. *J. ACM*, 49(6):753–784 (electronic), 2002.

[SS71] Doz Dr A Schönhage and Volker Strassen. Schnelle multiplikation grosser zahlen. *Computing*, 7(3-4):281–292, 1971.

[Tho] Mikkel Thorup. Undirected single-source shortest paths in linear time. http://www.diku.dk/PATH05/Thorup-1.pdf.

[Tho99] Mikkel Thorup. Undirected single-source shortest paths with positive integer weights in linear time. *Journal of the ACM (JACM)*, 46(3):362–394, 1999.

[Too63] Andrei L Toom. The complexity of a scheme of functional elements realizing the multiplication of integers. In *Soviet Mathematics Doklady*, volume 3, pages 714–716, 1963.

[vM11] Dieter van Melkebeek. Lecture note 1 for University of Wisconsin-Madison Course CS 710 on Complexity Theory. http://pages.cs.wisc.edu/ dieter/Courses/2011f-CS710/Scribes/PDF/lecture01.pdf, 2011.

[Wei12] Oren Weimann. Lecture notes 14 for MIT Course 6.851 Advanced Algorithms. http://courses.csail.mit.edu/6.851/spring12/scribe/L14.pdf, 2012.

[Wil07] R Ryan Williams. *Algorithms and resource requirements for fundamental problems.* PhD thesis, Carnegie Mellon University, 2007.