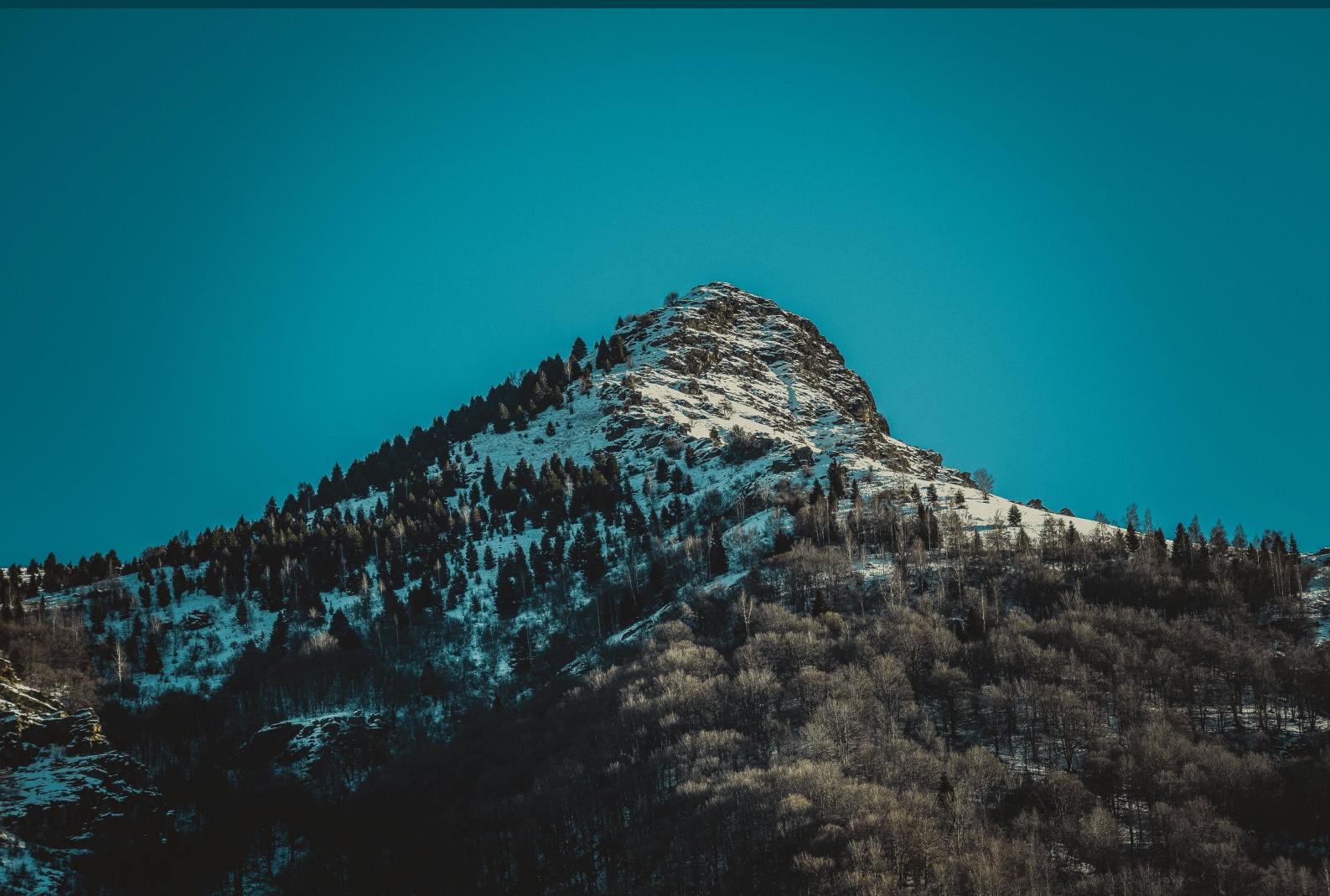


1st Edition

Analysis of Algorithms

Comprehensive Theory Notes & Workbook for UCS Students



UCSYSG
Academics

Yan Naing Soe

1st Edition

Analysis of Algorithms

Comprehensive Theory Notes & Workbook for UCS Students

Image Credit:

Cover photo: **Snow-Covered Mountain Under Blue Sky** taken by Goce Veselinovski at Dec 17, 2019 in Municipality of Bitola, North Macedonia. This symbolizes the height and challenge of problems that algorithms are made to solve. Great effort and consistency are required to study the algorithms just like the way it takes for one to get to the mountain.

(Source - <https://www.pexels.com/photo/snow-covered-mountain-under-blue-sky-4248116/>)



UCSYSG
Academics

Yan Naing Soe

Preface

Algorithm ဆိုတဲ့ စကားလုံးကို စာဖတ်သူတွေအနေနဲ့ ကွန်ပျူးတာတဗ္ဗာသို့လ် ဒုတိယနှစ်က တည်းက ကြေားဖူးပြီးသားဖြစ်ပါလိမ့်မယ်။ အခုတတိယနှစ်မှာတော့ algorithm တွေအကြောင်းကို အသေးစိတ် ခဲ့ခြမ်းစိတ်ဖြာပြီး လေ့လာကြရမှာ ဖြစ်ပါတယ်။ ဒီဘာသာရပ်ကို အလွယ်တကူနားလည် ကျေညာကိန်းဖို့အတွက် ဒုတိယနှစ်တုန်းက data structures and algorithm သဘောတရားတွေနဲ့ ရင်းနှီးနေဖို့ လိုပါမယ်။

ဒီဘာသာရပ်ကို စာရေးသူတို့ သင်ကြားရတုန်းက The Design and Analysis of Computer Algorithms စာအုပ်ထဲက Chapter 1 to 3 ကို သင်ခဲ့ရပါတယ်။ ဒါကြောင့် ဒီမှတ်စုစာအုပ်ထဲမှာ အဲဒီအခန်းသုံးခန်းထဲက သဘောတရားတွေကိုပဲ စုစဉ်းပေးထားပါတယ်။ ထူးခြားချက်အနေနဲ့ ဒီအခန်းသုံးခန်းထဲမှာ chapter 1 က အကြောင်းအရာအားလုံးရဲ့ တစ်ဝက်နှီးပါး လောက်ဖြစ်နေတာပါ။ ကျွန်ုတဲ့ chapter 2, 3 မှာက တွက်စရာ၊ ကျက်စရာ ဘာမှ သိပ်မရှိသလောက်ပါပဲ။

အရင်အတန်းတုန်းက လေ့လာခဲ့ရတဲ့ data structures and algorithms ဘာသာရပ်လို့ပဲ အခုလဲ trace တွေချည်း လက်မလည်အောင် လိုက်ရမှာဖြစ်ပါတယ်။ ဒါကြောင့် ချရေးလေ့ကျင့်ဖို့ မဖြစ်မနေ လိုအပ်ပါတယ်။ ဒု့အပြင် သဘောတရားကို တကယ်နားလည်မှပဲ ဘာတွေချရေးနေသလဲ၊ ဘာကြောင့် ဒါတွေလေ့လာနေရတယ်၊ ဘယ်နေရာမှာ အသုံးဝင်တယ်ဆိုတာကို သိနိုင်မှာဖြစ်တဲ့အတွက် textbook အပါအဝင် algorithm နဲ့ပတ်သက်တဲ့ စာအုပ်တွေဖတ်ဖို့ အတန်းမှန်မှန်တက်ဖို့ အင်မတန် အရေးကြီးပါတယ်။ ဒီလို လေ့လာမှပဲ tutorial တွေ၊ စာမေးပွဲတွေမှာ ကောင်းကောင်းမွန်မွန် ဖြေဆိုနိုင်မှာဖြစ်ပါတယ်။ အဲဒီလို လေ့လာရာမှာ ဒီမှတ်စုစာအုပ်လေးက အနည်းနဲ့အများ အထောက်အကူပြုပေးနိုင်မယ်လို့ မျှော်လင့်ပါတယ်။ ဒါဟာလဲ စာရေးသူရဲ့ အဓိကရည်ရွယ်ချက်ပါပဲ။ ဒီဘာသာရပ်ကို သင်ကြားပေးခဲ့တဲ့ ရန်ကုန်ကွန်ပျူးတာတဗ္ဗာသို့လ်၊ ကွန်ပျူးတာသိပ်မဟာငွာနာက တို့ချယ်အော်အေးမော်ကိုလဲ ဒီစာအုပ်လေးနဲ့ ဂါရဝါပြုလိုက်ပါတယ်။

Table of Contents

Chapter 1 Models of Computation.....	1
1.1 Algorithms and their Complexity	1
1.2 Random Access Machines	2
1.3 Computational Complexity of RAM Programs	5
1.5 Abstractions of RAM.....	12
1.6 The Turing Machine	16
Chapter 2 Design of Efficient Algorithms	23
2.1 Divide and Conquer Approach	23
2.2 Dynamic Programming.....	26
Chapter 3 Sorting and Order Statistics	30
3.1 Radix Sort	30
3.2 Heapsort	33
3.3 Quicksort.....	35
3.4 Order Statistics.....	37

Chapter 1

Models of Computation

Problem တစ်ခုအတွက် အကောင်းဆုံးသော solution ဟာဘာလဲ၊ ဘယ် algorithm တွေသုံးလို ရမလဲ၊ ရလဒ်ချင်းတူရင်တောင် algorithm တစ်ခုနဲ့တစ်ခု performance ချင်း ဘယ်လိုနိုင်းယူဉ်မလဲ စတဲ့ ဓမ္မနှင့်တွေဟာ ကွန်ပျူးတာသို့လေ့လာသူတွေအတွက်ရော၊ programmer တွေအတွက်ပါ စိတ်ဝင်စားစရာ အကြောင်းအရာတွေ ဖြစ်ပါတယ်။ ဒီအခန်းမှာတော့ algorithm အကြောင်း၊ algorithm တွေရဲ့ complexity တွေအကြောင်း၊ ဒီလို complexity တွေကို တိုင်းတာရာမှာ အသုံးဝင်တဲ့ computation model တွေ အကြောင်း ဆွေးနွေးတင်ပြသွားမှာ ဖြစ်ပါတယ်။

1.1 Algorithms and their Complexity

Problem တစ်ခု ဖြေရှင်းပုံဖြေရှင်းနည်းအဆင့်ဆင့်ကို algorithm လိုပေါ်ပါတယ်။ တစ်နည်း computer program တွေရဲ့ design ကိုလဲ algorithm လိုပေါ်ပါတယ်။ ဒီ algorithm တွေမှာ problem တွေ ကို ဖြေရှင်းဖို့ လိုအပ်တဲ့ resource တွေကို complexity လိုပေါ်ပါ တယ်။ တစ်နည်းအားဖြင့် problem တစ်ခု ကို ဖြေရှင်းဖို့ လိုအပ်တဲ့ အချိန်ပမာဏကို time complexity လိုပေါ်ပြီး problem တစ်ခုဖြေရှင်းဖို့ လိုအပ်တဲ့ computer memory ပမာဏကို space complexity လိုပေါ်ပါတယ်။ Problem size ကြီးလာတာနဲ့အမျှ ပြောင်းလဲသွားတဲ့ complexity behavior ကိုတော့ asymptotic complexity လိုပေါ်ပါတယ်။

Algorithm တွေကိုကြည့်ပြီး သူတို့ရဲ့ complexity တွေကို တွက်ထုတ်နိုင်သလို complexity ကိုကြည့်ပြီးတော့လဲ algorithm တစ်ခုက အချိန်အတိုင်းအတာတစ်ခုအတွင်း ကိုင်တွယ်နိုင်တဲ့ problem size ကို တွက်ထုတ်နိုင်ပါတယ်။ ဒီလိုတွက်ထုတ်ရာမှာ 1 second = 1000 times လို့ ယူမှတ်ပြီး algorithm တွေ တစ်စတုရန်အတွင်း လက်ခံနိုင်တဲ့ input ပမာဏကို ရှာတာဖြစ်ပါတယ်။ ဥပမာအနေနဲ့ Example 1.1 ကိုကြည့်ပါ။

Example 1.1. Calculate the input size of an algorithm which have time complexity $T(n) = n \log n$, assuming 1000 times equals 1 second.

Solution:

$$\begin{aligned}
 T(n) &= n \log n \\
 n \text{ input} &= n \log n \text{ time} \\
 n \log n \text{ time} &= n \text{ inputs} \\
 e^{n \log n} \text{ time} &= e^n \text{ inputs} \\
 n^n \text{ time} &= e^n \text{ inputs} \\
 5^5 \text{ time} &= e^5 \text{ inputs} \\
 1 \text{ second} &= e^5 \text{ inputs} \approx 148.41 \text{ inputs}
 \end{aligned}$$

Exercise 1.1: Input Size Calculation

Calculate the input sizes of an algorithm of complexity:

- i. n
- ii. n^2
- iii. n^3

1.2 Random Access Machines

Random Access Machine ဆိတာဟာ accumulator တစ်ခုတည်းပါတဲ့ ကွန်ပျူးတာတစ်မျိုး ဖြစ်ပါတယ်။ ဒီ machine အပါအဝင် RASP, Turing machine တွေကို computing model အဖြစ်သာ အသုံးပြုပြီး ကိန်းဂဏ်နှုန်းတွက်ချက်ဖို့ အကွရာနဲ့ပတ်သက်တဲ့ operation တွေလုပ်ဖို့ပဲ သုံးပါတယ်။

RAM တစ်ခုမှာ input, output tape, memory နဲ့ program တွေပါဝင်ပါတယ်။ Input tape, output tape တွေမှာ integer တစ်လုံးစာ ဆုံးတဲ့ စတုရန်းကွက်လေးတွေ ပါဝင်ပြီး tape head တစ်ခုစီနဲ့ ထောက်ထားပါတယ်။ RAM အလုပ်လုပ်တဲ့ အခါ input tape ထဲရောက်နေတဲ့ symbol တွေကို tape head နဲ့ ဖတ်ယူတာဖြစ်ပြီး output ထုတ်ပြစ်ရာရှိတဲ့ အခါ output tape ရဲ့ tape head ထောက်နေတဲ့ အကွက်ထဲ output ထုတ်ပေးတာပါ။ RAM ရဲ့ program ဟာ memory ထဲမှာမဟုတ်ဘဲ သီးခြားတည်ရှုပြီး ကိုယ့်ကိုယ် ဖြင့်လို့မရတဲ့ instruction တွေ ပါဝင်ပါတယ်။ Memory ထဲမှာတော့ integer တစ်လုံးစာဆုံးတဲ့ register တွေ ပါဝင်ပါတယ်။ Memory ရဲ့ ပထမဆုံး register ကို accumulator လို့ခေါ်ပါတယ်။ RAM ကပြုလုပ်တဲ့ သချာဆိုင်ရာတွက်ချက်၊ နှိုင်းယူလှမှ အကုန်လုံးကို ဒီတစ်နေရာတည်းမှာ စုပြုပြုလုပ်တာဖြစ်ပြီး integer တစ်လုံးပဲ ဆုံးပါတယ်။ RAM တစ်ခုရဲ့ တည်ဆောက်ပုံကို Figure 1.1 မှာ ကြည့်နိုင်ပါတယ်။

RAM program မှာပါတဲ့ instruction တွေဟာ label ထိုးထားတဲ့ အစဉ်အတိုင်း ရှိနေပြီး instruction တစ်ခုကြောင်းစီမှာ operation code အပိုင်းနဲ့ operand ပိုင်း ဆိုတဲ့ အစိတ်အပိုင်းနှစ်ခု ပါပါတယ်။ Operand နေရာမှာ ကိန်းသေတန်ဖိုး($= i$)၊ register နံပါတ်နဲ့ (i) indirect addressing ($* i$) တွေ အသုံးပြုနိုင်ပါတယ်။

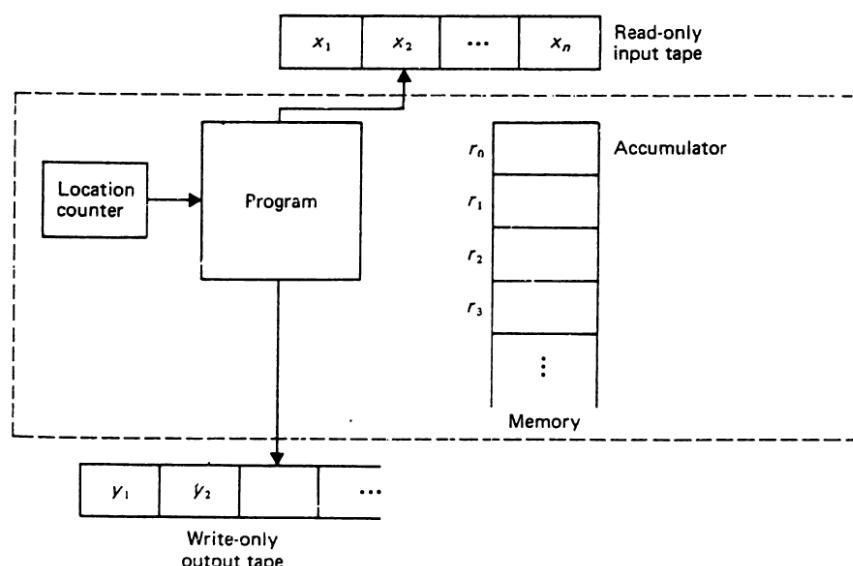


Figure 1.1 RAM တည်ဆောက်ပုံ

RAM instruction အပြည့်အစုံရယ်၊ သူတို့ရဲ့ အဓိပ္ပာယ်တွေရယ်ကို Figure 1.2 မှာ ဖော်ပြထားပါတယ်။ ဒီ instruction တွေထဲမှာ သချာဆိုင်ရာ၊ input/output ဆိုင်ရာ၊ indirect addressing ဆိုင်ရာ၊ branching ဆိုင်ရာ instruction တွေ ပါဝင်ပါတယ်။ သူတို့ရဲ့ အလုပ်လုပ်ပုံကို ဖော်ပြဖို့အတွက် value of ဆိုတဲ့ function နဲ့ contents of ဆိုတဲ့ function နှစ်ခုကို အသုံးပြုသွားမှာဖြစ်ပြီး value of ဆိုတာသည်

operand တွေရဲ့ တန်ဖိုး၊ contents of ဆိုတာသည် register တစ်ခုထဲ ဝင်နေဖို့ တန်ဖိုးကို အသီးသီး ကိုယ် စားပြုပါတယ်။ သူတို့နှစ်ခုရဲ့ ဆက်သွယ်ချက်ကတော့ အောက်ပါအတိုင်း ဖြစ်ပါတယ်။

$$\begin{aligned} v(< i) &= i, \\ v(i) &= c(i), \\ v(* i) &= c(c(i)) \end{aligned}$$

Instruction	Meaning
1. LOAD a	$c(0) \leftarrow v(a)$
2. STORE i	$c(i) \leftarrow c(0)$
STORE $*i$	$c(c(i)) \leftarrow c(0)$
3. ADD a	$c(0) \leftarrow c(0) + v(a)$
4. SUB a	$c(0) \leftarrow c(0) - v(a)$
5. MULT a	$c(0) \leftarrow c(0) \times v(a)$
6. DIV a	$c(0) \leftarrow \lfloor c(0)/v(a) \rfloor \dagger$
7. READ i	$c(i) \leftarrow \text{current input symbol.}$
READ $*i$	$c(c(i)) \leftarrow \text{current input symbol. The input tape head moves one square right in either case.}$
8. WRITE a	$v(a)$ is printed on the square of the output tape currently under the output tape head. Then the tape head is moved one square right.
9. JUMP b	The location counter is set to the instruction labeled b .
10. JGTZ b	The location counter is set to the instruction labeled b if $c(0) > 0$; otherwise, the location counter is set to the next instruction.
11. JZERO b	The location counter is set to the instruction labeled b if $c(0) = 0$; otherwise, the location counter is set to the next instruction.
12. HALT	Execution ceases.

Figure 1.2 RAM instruction (၁၂) ခုရဲ့ အဓိပ္ပာယ်

Tip 1.1: RAM Program ရေးနည်း

1. RAM program က တောင်းဆိုတဲ့ input, output တွေကို စဉ်းစားပါ။ Memory map ဆဲပါ။
2. Sample input တစ်ခု ကိုစဉ်းစားပြီး အဲဒီ input ကို ထုတ်ပေးရမယ့် output ဖြစ်အောင် ဘယ်လိုတွက် ရမလဲ ဆိုတာကို အကြမ်းရေးခြစ်ပြီး စဉ်းစားပါ။
3. ပုစ္စာက သတ်မှတ်ထားတဲ့ complexity ရှိနေရင် အဲဒီ complexity အတိုင်း ရအောင် စဉ်းစားပါ။
4. စဉ်းစားလိုရလာတဲ့ program အကြမ်းကို Pidgin ALGOL ပုံစံနဲ့ ချေရေးပါ။ (Pidgin ALGOL ဆိုတာဟာ conceptual algorithm တွေကို ဖော်ပြနိုင်တဲ့ language တစ်ခုဖြစ်ပြီး ဒီအကြောင်းကို referenced textbook ရဲ့ Chapter 1, section 1.8 မှာ ဖတ်နိုင်ပါတယ်။)
5. Pidgin ALGOL ကိုကြည့်ပြီး RAM program အဖြစ်ပြောင်းပါ။

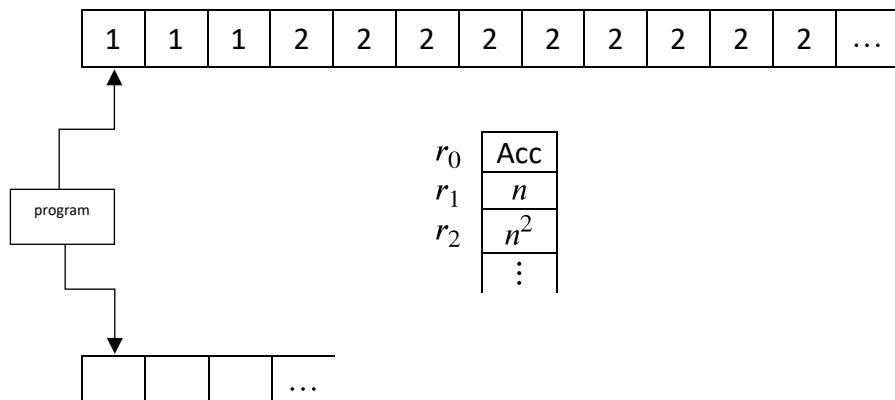
Caution: RAM program ရေးရာမှာ while condition တွေ စစ်တဲ့အခါကျ JUMP instruction ကနေပဲ တစ်ဆင့် HALT လုပ်သင့်ပါတယ်။ ဖြစ်နိုင်သရွှေ့ HALT instruction ကို program ရဲ့ နောက်ဆုံးမှာ ရေးသင့် ပါတယ်။

Example 1.2. Give a Pidgin ALGOL and RAM program which accepts inputs of the form $1^n 2^{n^2} 0$.

Solution:

Input - $1^n 2^{n^2} 0$, Output – accept (1) or not accept (0)

For $n = 3$, the input would be 111 222 222 222 0.



Pidgin ALGOL

```

begin
  read r1;
  if  $r_1 = 0$  then write 0
  else
    begin
       $r_2 \leftarrow 0$ ;
      while  $r_1 = 1$  do
        begin
           $r_2 \leftarrow r_2 + 1$ ;
          read  $r_1$ ;
        end
       $r_2 \leftarrow r_2 * r_2$ ;
      while  $r_1 = 2$  do
        begin
           $r_2 \leftarrow r_2 - 1$ ;
          read  $r_1$ ;
        end
      if  $r_1 = 0$  and  $r_2 = 0$  then write 1
      else write 0;
    end
  end
end

```

RAM program

```

    READ  1
    LOAD  1
    JZERO if
    JUMP  else
else:   LOAD = 0
        STORE 2
while1: LOAD 1
        SUB   = 1
        JZERO continue1
        JUMP  endwhile1
continue1: LOAD 2
        ADD   = 1
        STORE 2
        JUMP  while1
endwhile1: LOAD 2
        MULT  2
        STORE 2
while2:  LOAD 1

```

LOAD 2	SUB = 2
JZERO continue2	JUMP endwhile2
STORE 2	LOAD 2
READ 1	SUB = 1
JUMP while2	STORE 2
LOAD 1	READ 1
JZERO extend	JUMP if
JUMP if	LOAD 2
LOAD 2	JZERO accept
MULT 2	JUMP if
STORE 2	WRITE = 1
LOAD 1	HALT
	WRITE = 0
	HALT

1.3 Computational Complexity of RAM Programs

RAM program တွေရဲ complexity နဲ့ ပတ်သက်လာရင် worst-case complexity, average-case complexity ဆိုပြီး နှစ်မျိုးရှိနိုင်ပါတယ်။ Worst-case algorithm တစ်ခု လက်ခံနိုင်တဲ့ input ပမာဏ၊ အများဆုံးကို ကိုင်တွယ်ရတဲ့ အခြေအနေကို ရည်ညွှန်းပြီး average-case ကတော့ ပျမ်းမျှ input ပမာဏနဲ့ အလုပ်လုပ်တဲ့ အခြေအနေကို ဆိုလိုပါတယ်။

RAM program တစ်ခုရဲ complexity တွေကို တိုင်းတာတဲ့ အခါမှာ uniform cost criteria ဆိုတဲ့ time 1 unit, 1 register စနစ်အရ ယေဘုယျအားဖြင့် တိုင်းတာနိုင်သလို algorithm ရဲ့ တကယ့် complexity အတိအကျကို ကိုယ်စားပြုနိုင်တဲ့ logarithmic cost criteria အားဖြင့်လဲ တိုင်းတာနိုင်ပါတယ်။ Logarithmic cost criteria $I(i)$ ကို အောက်ပါ function နဲ့ ကိုယ်စားပြုနိုင်ပါတယ်။

$$I(i) = \begin{cases} \lceil \log|i| + 1 \rceil, & i \neq 0 \\ 1, & i = 0 \end{cases}$$

RAM program တစ်ပုဒ်ကို စတွက်တော့မယ်ဆိုရင် ပုံစံဖြေရှင်းနည်း algorithm ကို Pidgin ALGOL နဲ့ အရင်ရေးပြီး ရေးထားတဲ့ algorithm မှာပါတဲ့ main operation တွေကို ဆွဲထွေတ်ရပါမယ်။

ဒီ main operation တွေဆိုတာဟာ ပေါင်း၊ နှုတ်၊ မြောက်၊ စား၊ စားကြံးရှာတာတွေအပြင် looping increment တိုးတာတွေလဲ ပါနိုင်ပါတယ်။

Tip 1.2: RAM Program Complexity ရာနည်း

1. Tip 1.1 ထဲက အဆင့်တွေအတိုင်း အကုန်ရေးပြီးသွားရင် Pidgin ALGOL ကိုကြည့်။ Main operation ကိုရှာ။ Loop တစ်ခုစီရဲ့ loop ပတ်တဲ့ အကြိမ်အရေအတွက်ကိုရှာ။
2. Uniform Cost (time complexity) အတွက် one unit of main operation takes one unit of time ဆိုတဲ့ ဆက်သွယ်ချက်ကနေ loop အကြိမ်အရေအတွက်အတိုင်း ဘယ်နဲ့ unit time ထွက်လာမလဲ ဆိုတာကိုတွေ့ကြုံ။ Big-O ယူ။
3. Uniform Cost (space complexity) အတွက် program မှာ သုံးခဲ့တဲ့ register အရေအတွက်ကိုယူ။ $O(1)$ ယူ။
4. Logarithmic cost (time complexity) အတွက် main operation ကို loop i အကြိမ်အထိ ပတ်ပြီး output formula ရတဲ့အထိ အဆင့်ဆင့်တွေ့ကြပြီ။ Log ယူ။ Total cost ဆိုပြီး ယူထားတဲ့ \log ကို summation 1 to (number of loop) အထိ ဖြန့်။ ဖြေရှင်း။ နောက်ဆုံးရလဒ်ကို Big-O ယူ။
5. Logarithmic cost (space complexity) အတွက် program မှာရခဲ့တဲ့ အကြိုးဆုံး ကိန်းတန်ဖိုးကို formula နဲ့ဖော်ပြု။ Log ယူ။ Big-O ယူ။

Example 1.3. Estimate the uniform cost and logarithmic costs of both time and space complexities for the RAM program in Example 1.2.

Solution:

Uniform Cost

(i) Time Complexity

One ADD/SUB instruction takes → 1 unit of time

$n + n^2$ ADD/SUB instructions take → $n + n^2$ unit of time ∴ $O(n^2)$

(ii) Space Complexity

Number of registers used = 3 ∴ $O(1)$

Logarithmic Cost

(i) Time Complexity

For 1st loop,

1st ADD instruction yields → $0 + 1 = 1$

2nd ADD instruction yields → $(0 + 1) + 1 = 2$

3rd ADD instruction yields → $(0 + 1 + 1) + 1 = 3$

i^{th} ADD instruction yields → $(0 + 1 + 1 + \dots + 1) + 1 = i$

By taking the log, we have $\log i$

Total cost is

$$\begin{aligned}
\sum_{i=1}^n \log i &= \log 1 + \log 2 + \log 3 + \dots + \log n \\
&= \log(n - (n - 1)) + \log(n - (n - 2)) + \log(n - (n - 3)) + \dots + \log(n - (n - n)) \\
&= \log n + \log n + \log n + \dots + \log n \\
&= n \log n \therefore O(n \log n)
\end{aligned}$$

For 2nd loop,

1st SUB instruction yields $\rightarrow n^2 - 1$

2nd SUB instruction yields $\rightarrow n^2 - 2$

3rd SUB instruction yields $\rightarrow n^2 - 3$

i^{th} SUB instruction yields $\rightarrow n^2 - i$

By taking the log, we have $\log(n^2 - i)$

Total cost is

$$\begin{aligned}
\sum_{i=1}^{n^2} \log(n^2 + 1) &= \log(n^2 - 1) + \log(n^2 - 2) + \log(n^2 - 3) + \dots + \log(n^2 - n^2) \\
&= \log n^2 + \log n^2 + \log n^2 + \dots + \log n^2 \\
&= n^2 \log n^2 = 2n^2 \log n \therefore O(n^2 \log n)
\end{aligned}$$

(ii) Space Complexity

Largest integer stored in memory = n^2

By taking the log, we have $\log n^2 = 2 \log n \therefore O(\log n)$

Exercise 1.2: RAM Programs

1. Give an $O(n)$ step RAM program which computes 2^{2^n} given n . What is the (a) uniform cost, (b) logarithmic cost of your program?
2. Write a RAM program of uniform cost time complexity $O(\log n)$ to compute n^n .
3. Give RAM program which computes $n!$ given input n .
4. Write RAM instructions for a program which computes $f(n)$ such that

$$f(n) = \begin{cases} n^n, & n \geq 1 \\ 0, & \text{otherwise.} \end{cases}$$

1.4 Stored Program Model

Random Access Stored Program ဆိုတာဟာ program ကို memory ထဲ ထည့်သိမ်းထားနိုင်တဲ့ computing model တစ်ခုဖြစ်ပါတယ်။ ဒါ model မှာပါဝင်တဲ့ အစိတ်အပိုင်းတွေဟာ RAM နဲ့ အတူတူပဲဖြစ်ပြီး program မှာပါတဲ့ instruction တွေဟာ သူတို့ကိုယ်သူတို့ ပြင်ဆင်ခွင့်ရှိတာ၊ indirect addressing ကို ခွင့်ပြု မထားတာတွေပဲ ကွာပါတယ်။

RASP program instruction တစ်ကြောင်းစီကို computer memory ထဲမှာ register နှစ်ခုစာနဲ့သိမ်းဆည်းထားပြီး operation code ရဲ့ encoding အတွက်တစ်ခန်း၊ operand အတွက်တစ်ခန်းဆိုပြီး နေရာ ယူပါတယ်။ Operation code တွေကို စာသားနာမည်နဲ့ မဟုတ်တော့ဘဲ encoding နဲ့ အသုံးပြုလာမှာ ဖြစ်ပြီး Figure 1.3 မှာ လေ့လာနိုင်ပါတယ်။

Instruction	Encoding	Instruction	Encoding
LOAD i	1	DIV i	10
LOAD $=i$	2	DIV $=i$	11
STORE i	3	READ i	12
ADD i	4	WRITE i	13
ADD $=i$	5	WRITE $=i$	14
SUB i	6	JUMP i	15
SUB $=i$	7	JGTZ i	16
MULT i	8	JZERO i	17
MULT $=i$	9	HALT	18

Figure 1.3 RASP instruction တွေရဲ့ encoding ပေါ်

RASP နဲ့ ရှုံးမှာလေ့လာခဲ့တဲ့ RAM ဟာ ဆက်နှယ်နေပါတယ်။ အဲဒီ ဆက်စပ်မှုကို Theorem နှစ်ခုနဲ့ဖော်ပြနိုင်ပါတယ်။ ပထမသီအိုရမ်အရ RAM instruction တစ်ကြောင်းစီရဲ့ time complexity $T(n)$ ဟာ RASP instruction တစ်ကြောင်းစီရဲ့ time complexity $T(n)$ ရဲ့ k ဆ ရှိပါတယ်။

$$T(n)_{RAM} = kT(n)_{RASP}, \quad k = \text{any constant}$$

ဒုတိယသီအိုရမ်အရ RASP instruction တစ်ကြောင်းစီရဲ့ time complexity $T(n)$ ဟာ RAM instruction တစ်ကြောင်းစီရဲ့ time complexity $T(n)$ ရဲ့ k ဆ ရှိပါတယ်။

$$T(n)_{RASP} = kT(n)_{RAM}, \quad k = \text{any constant}$$

ဒီသီအိုရမ်နှစ်ခုအရ simulation ဆိုတဲ့ RAM program ကို RASP model နဲ့ ဖော်ပြတာ၊ RASP program ကို RAM model နဲ့ဖော်ပြတာတွေကို ပြုလုပ်နိုင်ပါတယ်။ ပထမသီအိုရမ်အရ RAM to RASP simulation လုပ်နိုင်ပြီး ဒုတိယသီအိုရမ်အရ RASP to RAM simulation လုပ်နိုင်ပါတယ်။

RAM to RASP Simulation

ပထမသီအိုရမ်အရ RAM to RASP simulation လုပ်ရာမှာ ကိုယ် simulate လုပ်ချင်တဲ့ program ကို သာမန် RAM program memory map ဆွဲပေးရပါတယ်။ တပြုင်တည်းမှာ RASP ရဲ့ memory ကိုလဲ Figure 1.4 ကလို ဆွဲပေးရပါမယ်။

RAM to RASP simulation လုပ်ရာမှာ ပထမဆုံး RAM accumulator ထဲက တန်ဖိုးတွေကို RASP register number 1 ထဲထည့်။ RAM ရဲ့ တွေ့ကြား register တွေထဲဝင်နေတဲ့ တန်ဖိုးတွေကိုလဲ $r + i$ အခန်းတွေ ထဲထည့်။

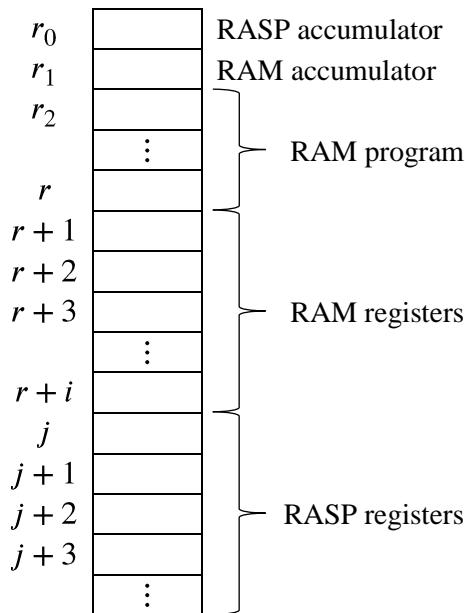


Figure 1.4 ပထမသီအိုရမ်အရ RASP memory ကိုခွဲပုံ

Register $r + i$ ထဲဝင်နေတဲ့ တန်ဖိုးတွေကို RASP accumulator ပေါ်ခွဲတင်ပြီး constant r ပေါင်း၊ RASP register တစ်နေရာရာမှာသိမ်း၊ သိမ်းထားတဲ့တန်ဖိုးကိုသုံးပြီး ကိုယ် simulation လုပ်ချင်တဲ့ instruction ကိုလုပ်။

RASP to RAM Simulation

ဒုတိယသီအိုရမ်အရ RASP to RAM simulation လုပ်ရာမှာလဲ ကိုယ် simulate လုပ်ချင်တဲ့ program ကို သာမန် RAM program memory map ခွဲပေးရပါတယ်။ အဲဒါကို RASP memory ပုံစံပြင်ပြီးမှ RAM memory map နောက်တစ်ခုထပ်ခွဲရပါတယ်။ ဒါကြောင့် ဒီ simulation မှာ memory map သုံးကြိမ် ခွဲရမှာဖြစ်ပါတယ်။ နောက်ဆုံးတစ်ခုကိုတော့ Figure 1.5 ကအတိုင်း ခွဲပေးရပါမယ်။

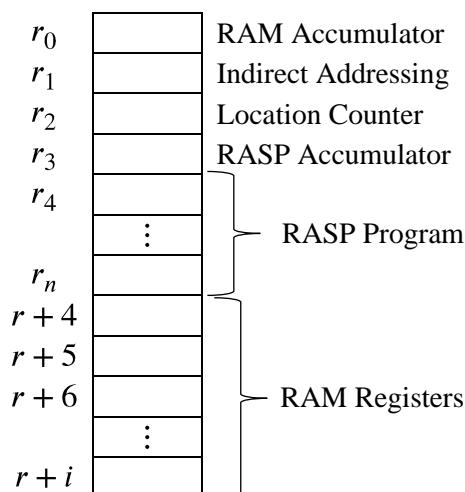
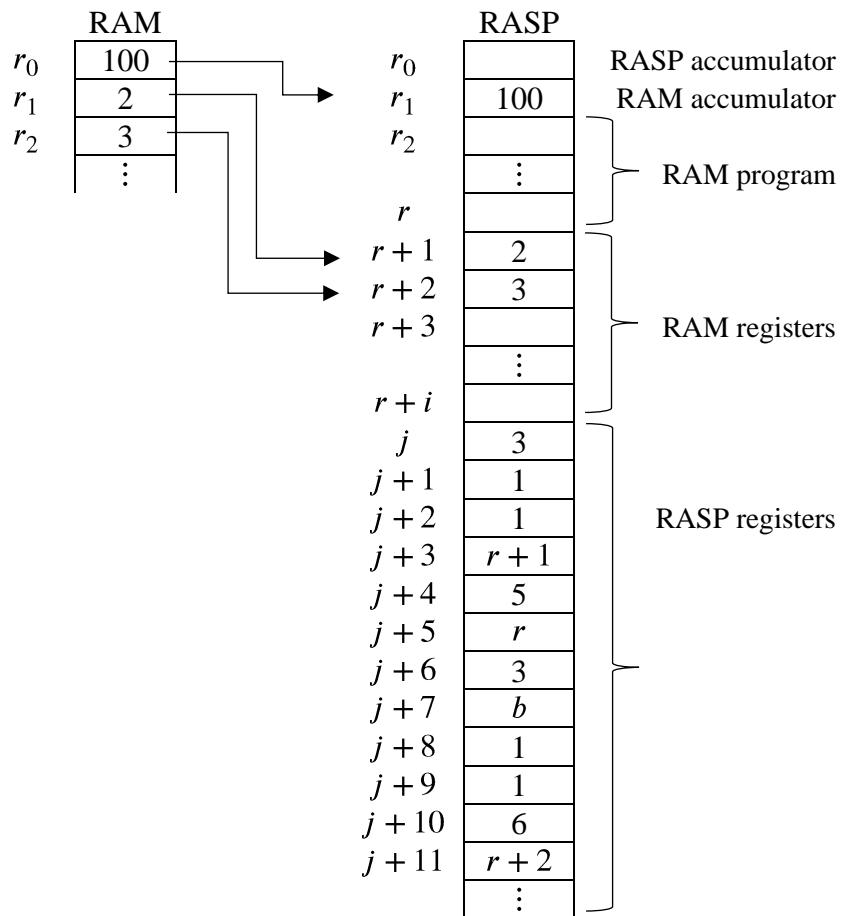


Figure 1.5 ဒုတိယသီအိုရမ်အရ RAM memory ကို ခွဲပုံ

RASP to RAM simulation လုပ်ရာမှာ register r_2 ဖြစ်တဲ့ location counter ကို တစ်တိုး။ Location counter ညွှန်ထားသော register ကိုဆွဲတင် ကိန်းသော 3 ပေါင်း၊ register r_1 ထဲထည့်။ ပြီးရင် ကိုယ် simulate လုပ်ချင်သော instruction ကိုလုပ်။ Location counter ကို တစ်ပြန်တိုးပြီး JUMP ခုနှစ်ခွဲဖို့ ထွေ။

Example 1.4. Simulate the RAM instruction SUB * i using RASP model. Assume there is a constant of value 100 in the RAM's accumulator. You can use any constant value to simulate the subtraction.

Solution:



RASP instruction for SUB * 1

```

STORE 1
LOAD  r + 1
ADD   = r
STORE j + 11
LOAD   1
SUB    j + 11

```

RASP instruction for SUB * i

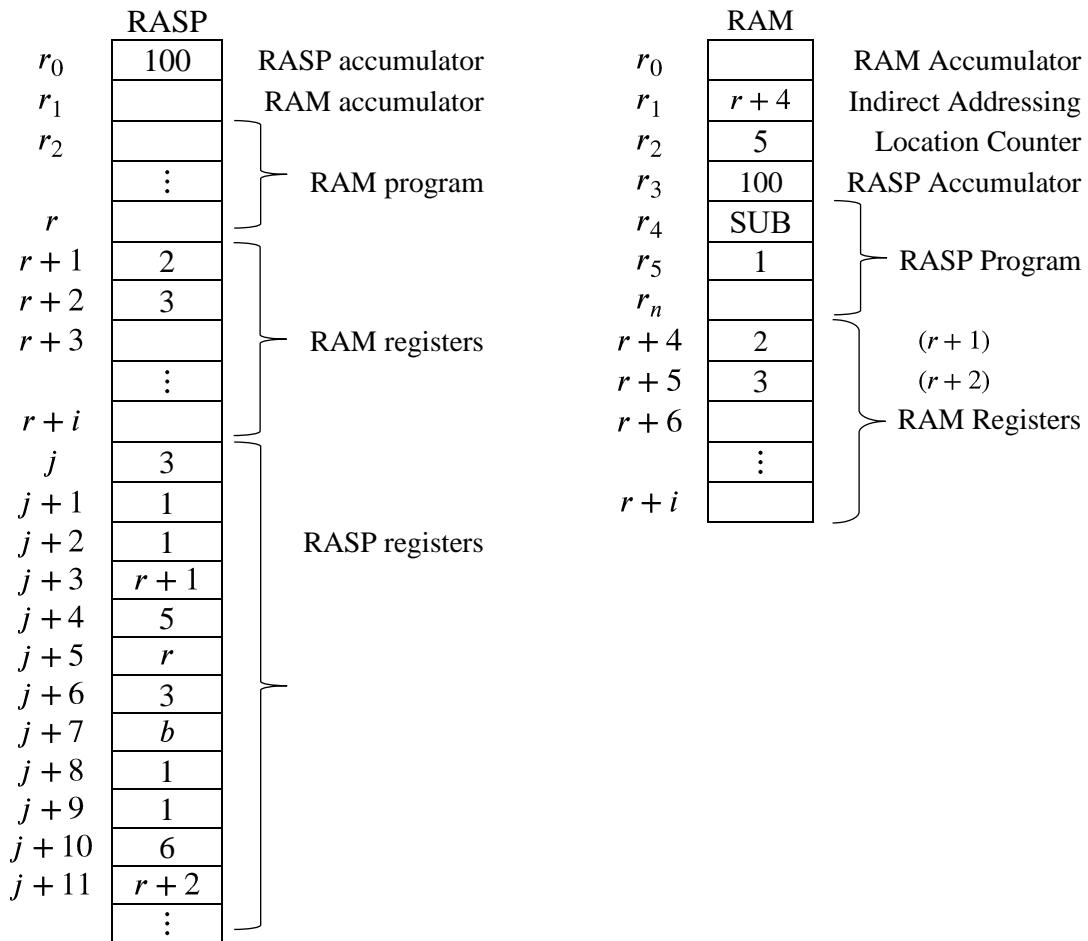
```

STORE 1
LOAD  r + i
ADD   = r
STORE j + 11
LOAD   1
SUB    j + 11

```

Example 1.5. Simulate the RASP instruction SUB i using RAM model. Assume there are already data exist as mentioned in Example 1.4.

Solution:



RASP instruction for SUB 1

```

LOAD 2
ADD = 1
STORE 2

LOAD * 2
ADD 3
STORE 1

LOAD 3
SUB * 1
STORE 3

LOAD 2
ADD = 1
STORE 2

JUMP b

```

RASP instruction for SUB i

```

LOAD 2
ADD = 1
STORE 2

LOAD * 2
ADD 3
STORE 1

LOAD 3
SUB * i
STORE 3

LOAD 2
ADD = 1
STORE 2

JUMP b

```

Exercise 1.3. RAM to RASP Simulations

Simulate the following RAM instructions in RASP.

- i. LOAD * i
- ii. STORE * i
- iii. MULT * i
- iv. DIV * i
- v. READ * i
- vi. WRITE * i

Exercise 1.4. RASP to RAM Simulations

Simulate the following RASP instructions in RAM.

- i. LOAD i
- ii. STORE i
- iii. ADD i
- iv. MULT i
- v. DIV i
- vi. READ i
- vii. WRITE i

1.5 Abstractions of RAM

ရှေ့မှာတွေခဲ့ကြတဲ့ RAM နဲ့ RASP တို့ဟာ လေးလာခဲ့ကြရတဲ့အတိုင်း ရိုးရှင်းတဲ့ problem လေးတွေ ဖြေရှင်းဖို့တောင် အင်မတန် ရှုပ်ထွေးလုပါတယ်။ တချို့ တွက်ချက်မှုတွေမှာ ဒီလိုရှုပ်ထွေးမှုကို ထည့်သုံးလို့ မရတဲ့အခါ RAM, RASP တဲ့က feature တချို့ကိုဖယ်ထားပြီး သုံးရပိုလွယ်ကူတဲ့ computing model တွေကို လိုအပ်လာပါတယ်။ ဒီအပိုင်းမှာတော့ RAM ကို abstract လုပ်ထားတဲ့ computing model သုံးခုအကြောင်း တင်ပြပေးသွားပါမယ်။

Straight-Line Programs

RAM program တွေရေးတုန်းက အလုပ်တစ်ခုခုကို ထပ်ခါတလဲလဲ လုပ်ဖို့လိုတဲ့ နေရာတွေမှာ looping တွေသုံးခဲ့ကြတာ မှတ်မိမှာပါ။ ဒီ computing model မှာတော့ problem တစ်ခုဖြေရှင်းဖို့အတွက် looping တွေအစား တကယ့် sequential instruction တွေကို အသုံးပြုသွားမှာဖြစ်ပါတယ်။ တစ်နည်းအား ဖြင့် looping တွေကို ဖြေချလိုက်တာဖြစ်ပါတယ်။ Straight-line program မှာ looping, condition နဲ့ဆိုင်တဲ့ branching instruction တွေ၊ indirect addressing တွေ၊ read instruction တွေကို ဖယ်ထုတ်ပစ်မှာ

Tip 1.3: Polynomial ပုံစံတွက်နည်း

1. ပေးထားတဲ့ polynomial ကို $n = 3$ အထိ ဖြန့်။
2. ဖြန့်ထားတဲ့ ကိန်းတန်းတွေကို x ပါဝါ တစ်ထပ်တည်းရှုတဲ့အထိ ဘုံထုတ်။
3. ဘုံထုတ်ထားတဲ့ expression တွေကို straight-line ပုံစံ အဆင့်ဆင့်ရေး။
4. Time complexity ကို ထွက်လာတဲ့ straight-line instruction အကြောင်းအရေအတွက်အတိုင်းယူ။
5. Space complexity ကိုတော့ သုံးလိုက်တဲ့ variable အရေအတွက်အတိုင်းယူ။

ဖြစ်ပြီး သူတို့အစား LOAD, STORE, WRITE, HALT instruction တွေ၊ ပေါင်းနှုတ်မြောက်စားတွေပဲ သုံးတော့ မှာဖြစ်ပါတယ်။ ဒါအပြင် တန်ဖိုးတွေကို သိမ်းဆည်းဖို့ register တွေအစား symbolic address တွေပဲ သုံးမှာ ဖြစ်ပါတယ်။ Straight-Line Program ရဲ့ ဥပမာတစ်ခုအနေနဲ့ polynomial ဥပမာကို လွှဲလာကြည့်ပါ။

Example 1.6. Write a straight-line program to evaluate the polynomial $p(x) = a_nx^n + a_{n-1}x^{n-1} + \dots + a_1x + a_0$ and find space and time complexities.

Solution:

$$p(x) = a_nx^n + a_{n-1}x^{n-1} + \dots + a_1x + a_0$$

$$\begin{aligned} n = 1, \quad & p(x) = a_1x + a_0 \\ n = 2, \quad & p(x) = a_2x^2 + a_1x + a_0 = (a_2x + a_1)x + a_0 \\ n = 3, \quad & p(x) = a_3x^3 + a_2x^2 + a_1x + a_0 = (a_3x^2 + a_2x + a_1)x + a_0 \\ & = ((a_3x + a_2)x + a_1)x + a_0 \end{aligned}$$

The straight-line program is

$n = 1,$	$t \leftarrow a_1 * x$	Space = 5
	$t \leftarrow t + a_0$	Time = 2
$n = 2,$	$t \leftarrow a_2 * x$	Space = 6
	$t \leftarrow t + a_1$	Time = 4
	$t \leftarrow t * x$	
	$t \leftarrow t + a_0$	
$n = 3,$	$t \leftarrow a_3 * x$	Space = 7
	$t \leftarrow t + a_2$	Time = 6
	$t \leftarrow t * x$	
	$t \leftarrow t + a_1$	
	$t \leftarrow t * x$	
	$t \leftarrow t + a_0$	

Time Complexity:

$$\begin{aligned} n = 1, \quad & O_A(n) = 2 = 2 \times 1 = 2n \\ n = 2, \quad & O_A(n) = 4 = 2 \times 2 = 2n \\ n = 3, \quad & O_A(n) = 6 = 2 \times 3 = 2n \\ \therefore O_A(n) &= O(2n) \end{aligned}$$

Space Complexity:

$$\begin{aligned} n = 1, \quad & O_A(n) = 5 = 4 + 1 = 4 + n \\ n = 2, \quad & O_A(n) = 6 = 4 + 2 = 4 + n \\ n = 3, \quad & O_A(n) = 7 = 4 + 3 = 4 + n \\ \therefore O_A(n) &= O(4 + n) \end{aligned}$$

Bitwise Operations

RAM program တွေကို bitwise operation တွေဖြစ်တဲ့ binary digit တွေ ပေါင်းခြင်း၊ မြောက်ခြင်း ပြုလုပ်ဖို့အတွက်လဲ သုံးနိုင်ပါတယ်။ ဒါ computation model က တကယ်တော့ straight-line program နဲ့

အတူတူပဲဖြစ်ပြီး variable တွေမှာ 0 နဲ့ 1 တန်ဖိုးနှစ်ခုပဲ သိမ်းနိုင်တာရယ်၊ ပေါင်းနှတ်မြောက်စား တွက်ချက်မှု တွေမဟုတ်ဘဲ logical operator တွေကိုသုံးတာရယ်ပဲ ကွာခြားပါတယ်။ Bitwise operation တွေသည် logic circuit တွေမှာ အသုံးဝင်ပါတယ်။

Bitwise operation တွေမှာ binary digit တွေကို ပေါင်းခိုင်းတဲ့ ပုစ္ဆာ၊ မြောက်ခိုင်းတဲ့ ပုစ္ဆာဆိုပြီး နှစ်ချိုး အခိုက်ရှိပါမယ်။ Binary digit အချင်းချင်းပေါင်းတဲ့ အခါ logical XOR operator ကိုသုံးပြီး မြောက်တဲ့ အခါကျေရင်တော့ logical AND operator ကို သုံးပါတယ်။ ဒီလို ပေါင်းတဲ့ မြောက်တဲ့ အခါမှာ carry တက်တဲ့ ကိစ္စကို မဖြစ်မနေ ထည့်စည်းစားပေးရပြီး carry တက်လာမယ့် digit နှစ်လုံးကို AND ပေးခြင်းအားဖြင့် carry ကို ကိုယ်စားပြုနိုင်ပါတယ်။

Tip 1.4: Bitwise Operation ပုစ္ဆာတွက်နည်း

1. ပေးထားတဲ့ binary digit နှစ်ခုကို လက်တန်း ချေပေါင်း/မြောက်။
2. အဖြေရလာတဲ့ digit တစ်လုံးစီကို သက်ဆိုင်ရာ ကိန်းလုံးနှစ်ခုရဲ့ ပေါင်းလဒ်/မြောက်လဒ်အဖြစ် ဖော်ပြု။ c_0, c_1, c_2 စသဖြင့် နာမည်ပေး။ Carry တက်မယ့် နေရာအတွက်လဲ တစ်လုံးအပိုထည့်။
3. ပေါင်းလဒ်/မြောက်လဒ်တွေကို logical operator တွေနဲ့ဖော်ပြု။ Carry တက်မယ့် case အတွက်လဲ ထည့်ရေး။
4. ရလာတာတွေကို straight-line program ရေး။ ရေးထားတဲ့ straight-line program ကိုကြည့်ပြီး circuit ဆဲ။

Example 1.6. Write a sequence of bit operations to compute the addition of a 3-bit integer $[a_2 a_1 a_0]$ and a 2-bit integer $[b_1 b_0]$ and draw the logic circuit.

Solution:

$$[a_2 a_1 a_0] + [b_1 b_0] = ?$$

$$\begin{array}{r}
 & a_2 & a_1 & a_0 \\
 + & b_1 & b_0 \\
 \hline
 c_3 & c_2 & c_1 & c_0
 \end{array}$$

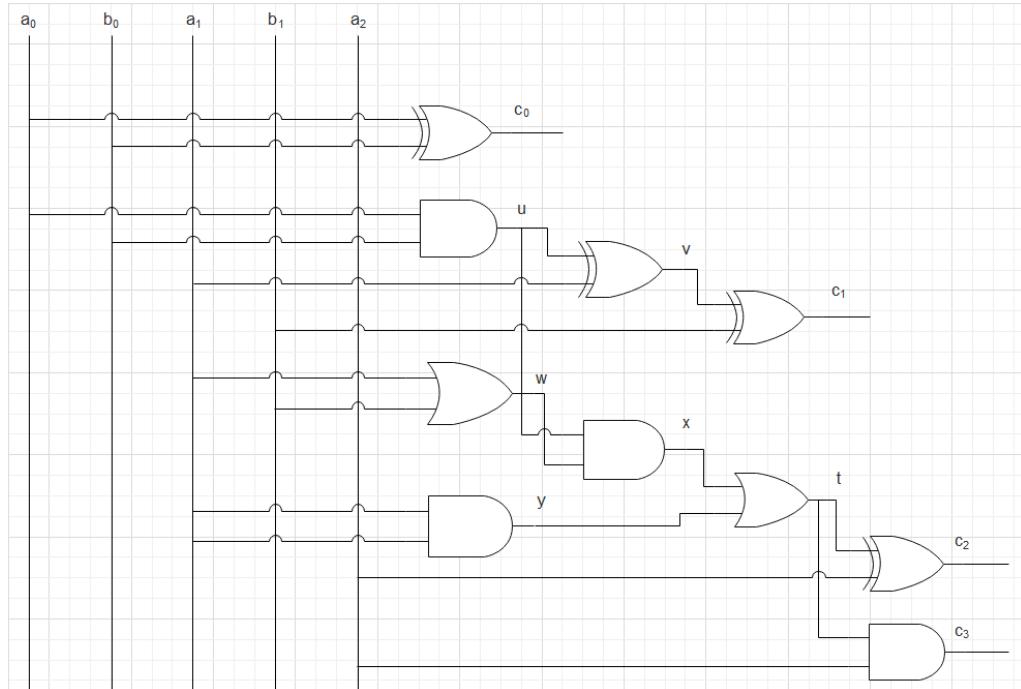
Bit operation is

$$\begin{aligned}
 c_0 &= a_0 \oplus b_0 \\
 c_1 &= ((a_0 \wedge b_0) \oplus a_1) \oplus b_1 \\
 c_2 &= ((a_0 \wedge b_0) \wedge (a_1 \vee b_1)) \vee (a_1 \wedge b_1) \oplus a_2 \\
 c_3 &= ((a_0 \wedge b_0) \wedge (a_1 \vee b_1)) \vee (a_1 \wedge b_1) \wedge a_2
 \end{aligned}$$

Bit program is

$$\begin{aligned}
 c_0 &\leftarrow a_0 \oplus b_0 \\
 u &\leftarrow a_0 \wedge b_0 \\
 v &\leftarrow u \oplus a_1
 \end{aligned}$$

$$\begin{aligned}
 c_1 &\leftarrow v \oplus b_1 \\
 w &\leftarrow a_1 \vee b_1 \\
 x &\leftarrow u \wedge w \\
 y &\leftarrow a_1 \wedge b_1 \\
 t &\leftarrow x \vee y \\
 c_2 &\leftarrow t \oplus a_2 \\
 c_3 &\leftarrow t \wedge a_2
 \end{aligned}$$



Exercise 1.5. Bitwise Operations

1. Write a sequence of bit operations to compute the addition of two 2-bit integers $[a_1a_0]$ and $[b_1b_0]$ and draw the logic circuit.
2. Write a sequence of bit operations to compute the multiplication of two 2-bit integers $[a_1a_0]$ and $[b_1b_0]$ and draw the logic circuit.

Decision Trees

RAM program တရှိမှာ branch instruction တွေ၊ တစ်နည်း condition စစ်ပြီး jump လုပ်ရတဲ့ နေရာတွေက အတော်ကိုရှုပေးနေတတ်ပါတယ်။ ဒီလို့နေရာတွေမှာ binary tree အမျိုးအစားတစ်မျိုးဖြစ်တဲ့

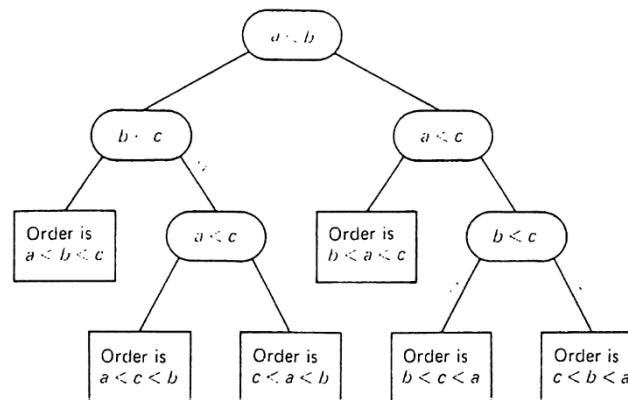


Figure 1.6
Decision Tree
ပြမာ

decision tree ကို အသုံးပြန်ပါတယ်။ ဒီ tree မှာပါတဲ့ parent node တွေဟာ condition တွေကို ကိုယ်စားပြုပြီး leaf node တွေကတော့ final output ကို ကိုယ်စားပြုပါတယ်။ ဥပမာအနေနဲ့ a, b, c ဆိုတဲ့ ကိန်းသုံးလုံးကို ငယ်စဉ်ကြိုးလိုက်စိတဲ့ decision tree ကို Figure 1.6 မှာကြည့်ပါ။ Decision tree ဆောက်တဲ့အခါ ဖြစ်နိုင်တဲ့ condition တွေကို node အဖြစ် ခွဲရေးပြီး final output တွေကိုတော့ သူ့အထက်က parent node အချင်းချင်း တိုက်ဆိုင် ပြီး ယူရပါမယ်။

1.6 The Turing Machine

RAM, RASP တွေလေ့လာလို့ ပြီးသွားတဲ့အခါ နောက်ဆုံး computing model ဖြစ်တဲ့ Turing Machine ကို လေ့လာကြပါမယ်။ Turing Machine ဆိုတာဟာ symbol လေးတွေကို process လုပ်ပေးတဲ့ ကွန်ပျုံတာတစ်မျိုးဖြစ်ပါတယ်။ Input အနေနဲ့ သူ့ဆီထည့်ပေးထားတဲ့ symbol တွေကို သူ့ကဖတ်ပြီး ကိုယ်လို ချင်တဲ့ output symbol တွေကို ချရေးပေးပါတယ်။ Turing machine တည်ဆောက်ပုံကို Figure 1.7 မှာ ကြည့်ပါ။

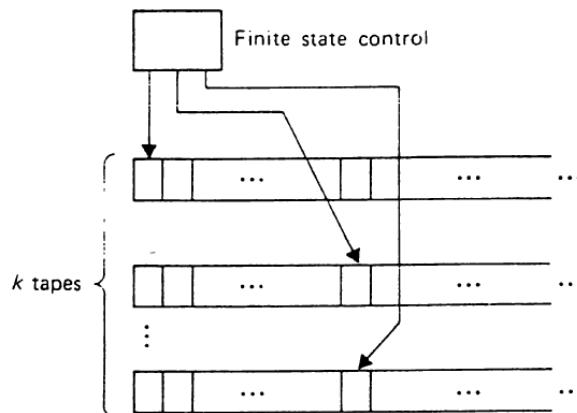


Figure 1.7 Multitape Turing machine တည်ဆောက်ပုံ

Multitape Turing Machine မှာ tape တွေများစွာ ပါဝင်ပြီး tape တစ်ခုစိမှာလဲ symbol တစ်ခုစာ ဆုံးတဲ့ စတုရန်းအတွက်လေးတွေပါရှိပါတယ်။ Tape တစ်ခုစိမှာ tape head လက်တံတွေ့ချောင်းစီပါဝင်ပြီး လက်ရှိ ရောက်နေတဲ့ cell ကွက်၊ နောက်သွားရမယ့် cell ကွက်စသေဖြင့် ထောက်ပြပေးပါတယ်။ ဒု့အပြင် tape head တွေဟာ ရှုံးရော နောက်ပါ လွတ်လပ်စွာ ရွှေ့နှင့်ပြီး read, write operation နှစ်ခုလုံးလဲ လုပ်နိုင်ပါတယ်။ Turing Machine ရဲ့ လူပ်ရှားတွက်ချက်မှုကို finite control ဆိုတဲ့ program ကြီးနဲ့ ထိန်းချုပ်ထားပါတယ်။ ဒီ machine ဟာ program run နေချိန်တောက်လျောက် state တစ်ခုမဟုတ် တစ်ခုမှာရှုံးနေပါတယ်။ State ဆို တာဟာ finite control ထဲက position ကို ဆိုလိုပါတယ်။ Turing machine ရဲ့ အလုပ်လုပ်ဆောင်သွားတာတွေကို instantaneous description အနေနဲ့လဲ ဖော်ပြန်ပါတယ်။

Turing Machine အနေနဲ့ symbol manipulation တွေ၊ language accepter တွေအဖြစ်နဲ့လဲ အသုံးပြန်ပါတယ်။ တစ်နည်း input symbol ပေးထားပြီး ကိုယ်လိုချင်တဲ့ ပြောင်းလဲ ခိုင်းတာတွေ၊ input symbol တစ်ခုဟာ ကိုယ်လိုချင်တဲ့ပုံစံကျမကျကို စစ်ဆေးတာတွေ လုပ်ပေးနိုင်ပါတယ်။

Turing Machine ပုစ္စာတွေကို တွက်နည်းအမျိုးမျိုးနဲ့ တွက်နိုင်ပါတယ်။ ဒီသင်ရှိုးအရတော့ အောက်က ဥပမာပုစ္စာမှာပါတဲ့ ပုံစံအတိုင်း တွက်ပြထားပါတယ်။ ပုစ္စာဖြေရှင်းနည်းဆိုပြီး အတိအကျပြောပြလို လဲ မလွယ်ကူပါဘူး။ ဒါပေမဲ့ ပုစ္စာအတော်များများကို တွက်လိုအဆင်ပြောမယ့် နည်းအဆင့်ဆင့်ကို Tip 1.5 မှာ ကြည့်နိုင်ပါတယ်။

Tip 1.5: Multitape Turing Machine ပုစ္စာတွေက်နည်း:

1. ပေးထားတဲ့ ပုစ္စာအမျိုးအစားကိုကြည့်ပြီး symbol manipulation လုပ်ခိုင်းတာလား၊ language accepter လုပ်ခိုင်းတာလားဆိုတာကို ခွဲပါ။ ပထမအမျိုးအစားဆိုရင် output တစ်ခု ထွက်အောင် တွက် ပေးရမှာဖြစ်ပြီး ဒုတိယအမျိုးအစားဆိုရင်တော့ accept state ရောက် မရောက် အဖြေထုတ်ပေးရပါ မယ်။
2. ပုစ္စာစတွက်ဖို့ sample input တစ်ခုရွေးပြီး အဲဒီ sample input အပေါ်မူတည်လို ပုစ္စာတွေက်ခိုင်းတဲ့ အတိုင်း စဉ်းစားပါ။
3. လိုအပ်မယ့် tape အရေအတွက်နဲ့ state တွေကို ဆက်စပ်စဉ်းစားပြီး state diagram ခွဲပါ။ ဆွဲတဲ့နေရာမှာ လက်တံ tape head အနေအထားကို ထည့်ဆွဲပါ။
4. Next-move function (ခေါ်) finite control ကို table ချုဆွဲပါ။
5. Instantaneous description ချေရေးပါ။
6. Complexity တွက်ပါ။ Time complexity အတွက် state တစ်ခုစီမှာ တွေ့ခဲ့တဲ့ tape head move အရေအတွက် စုစုပေါင်းကို ယူပါ။ Space complexity အတွက်ဆိုရင် special character အတွက် တစ်နေရာ၊ blank အတွက်တစ်နေရာ၊ sample input အတွက်လိုအပ်တဲ့ square အရေအတွက်ကို ပေါင်းပြီးယူပါ။

Example 1.7. Give a Turing machine which recognizes palindromes on the alphabet {0, 1}.

Solution:

Choose 11011 as a sample input.

q_0	\downarrow								
t_1	1	1	0	1	1	...			
t_2	\times	b				...			

q_2		\downarrow							
t_1	1	1	0	1	1	b			
t_2	\times	1	1	0	1	1	...		

q_1		\downarrow							
t_1	1	1	0	1	1	b			
t_2	\times	1	1	0	1	1	b		

q_3		\downarrow							
t_1	1	1	0	1	1	...			
t_2	\times	1	1	0	1	1	...		

q_4		↓					
t_1	1	1	0	1	1	...	
		↓					
t_2	×	1	1	0	1	1	...

The next move function is

Current State	Current Symbol on		Next Symbol on		Next State
	t_1	t_1	t_1	t_1	
q_0	1	b	$1, S$	\times, R	q_1
q_1	1	b	$1, R$	$1, R$	q_1
	0	b	$0, R$	$0, R$	q_1
	b	b	b, S	b, L	q_2
q_2	b	1	b, S	$1, L$	q_2
	b	0	b, S	$0, L$	q_2
	\times		b, L	\times, R	q_3
q_3	1	1	$1, S$	$1, R$	q_4
	0	0	$0, S$	$0, R$	q_4
q_4	1	1	$1, L$	$1, S$	q_3
	0	1	$0, L$	$1, S$	q_3
	1	0	$1, L$	$0, S$	q_3
	b		$1, S$	b, S	q_5
q_5					Accept

The instantaneous description is -

$$\begin{aligned}
 (q_0 11011, \quad q_0 b) &\vdash (q_1 11011, \quad \times q_1 b) \\
 &\vdash (1q_1 1011, \quad \times 1q_1 b) \\
 &\vdash (11q_1 011, \quad \times 11q_1 b) \\
 &\vdash (110q_1 11, \quad \times 110q_1 b) \\
 &\vdash (1101q_1 1, \quad \times 1101q_1 b) \\
 &\vdash (11011q_2 b, \quad \times 1101q_2 1b) \\
 &\vdash (11011q_2 b, \quad \times 110q_2 11b) \\
 &\vdash (11011q_2 b, \quad \times 11q_2 011b) \\
 &\vdash (11011q_2 b, \quad \times 1q_2 1011b) \\
 &\vdash (11011q_2 b, \quad \times q_2 11011b) \\
 &\vdash (1101q_3 1, \quad \times q_3 11011b) \\
 &\vdash (1101q_4 1, \quad \times 1q_4 1011b) \\
 &\vdash (110q_3 11, \quad \times 1q_3 1011b) \\
 &\vdash (110q_4 11, \quad \times 11q_4 011b) \\
 &\vdash (11q_3 011, \quad \times 11q_3 011b) \\
 &\vdash (11q_4 011, \quad \times 110q_4 11b) \\
 &\vdash (1q_3 1011, \quad \times 110q_3 11b) \\
 &\vdash (1q_4 1011, \quad \times 1101q_4 1b) \\
 &\vdash (q_3 11011, \quad \times 1101q_3 1b) \\
 &\vdash (q_4 11011, \quad \times 11011q_4 b) \\
 &\vdash (q_5 11011, \quad \times 11011q_5 b)
 \end{aligned}$$

Time Complexity, $T(n)$ is

State	Number of move
q_0	1
q_1	$n + 1$
q_2	$n + 1$
$q_3 + q_4$	$2n - 1$
q_5	1
Total	$4n + 3$
	$\therefore T(n) = 4n + 3$

Space Complexity is $S(n) = n + 2$

Exercise 1.6. Turing Machines

1. Specify a Turing machine which when given two binary integers on tape 1 and 2 will print their sum on tape 3. You may assume the left ends of the tapes are marked by a special symbol #.
2. Give a Turing machine which prints 0^{n^2} on tape 2 when started with 0^n on tape 1.
3. Give a Turing machine which accepts inputs of the form $0^n 1 0^{n^2}$.

Definitions

Algorithm

Algorithm is the design of computer programs.

Problem Size

The measure of the quantity of input data.

Time / Space Complexity

The time/space needed by an algorithm expressed as a function of the size of a problem is called the time/space complexity of the algorithm.

Asymptotic time / space complexity

The limiting behavior of the complexity as size increases is called the asymptotic time/space complexity.

Random Access Machine

A random access machine is a one-accumulator computer in which instructions are not permitted to modify themselves.

Accumulator

Accumulator is the first register of random access machine in which all computations take place.

Input tape

The input tape is a sequence of squares, each of which holds an integer.

Output tape

The output tape is a sequence of squares, which are initially blank but integers will be printed on when a WRITE instruction is executed.

Worst-Case time/space complexity

The worst-case time/space complexity of a RAM program is the function $f(n)$ which is the maximum, over all inputs of size n , of the sum of the time/space taken by each instruction executed.

Expected time/space complexity

The expected time/space complexity is the average over all inputs of size n , of the same sum.

Uniform Cost Criterion

Under the uniform cost criterion, each RAM instruction requires one unit of time and each register requires one unit of space.

Logarithmic Cost Criterion

Logarithmic cost criterion is the limited size of a real memory word.

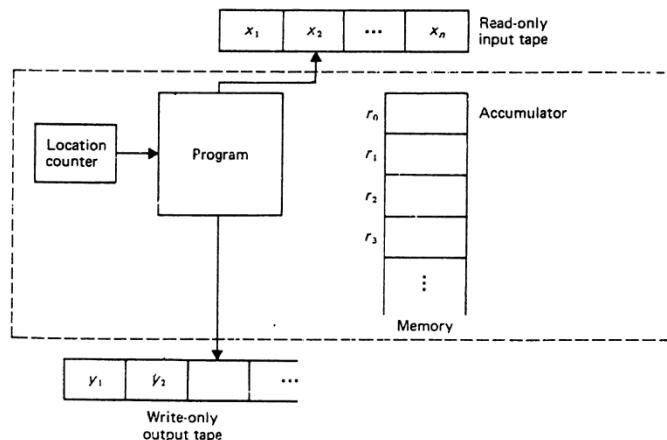
Random Access Stored Program Model

A random access stored program model is similar to RAM with the exception that the program is in memory and can modify itself.

Conceptual Notes

1. Structure of Random Access Machine

A random access machine is a one-accumulator computer in which instructions are not permitted to modify themselves. A RAM consists of a read-only input tape, a write-only output tape, a program and a memory. The read-only input tape is a sequence of squares, each of which holds an integer. A symbol is read from the input tape as the tape head moves one square to the right. Write only output tape is also a sequence of squares which are initially blank. When a WRITE instruction is executed, an integer is printed on a square pointed by the output tape head. And then the tape head is moved one square to the right. RAM memory consists of a sequence of registers $r_0, r_1, r_2, \dots, r_n$, each of which is capable of holding an integer of arbitrary size. We place no upper bound on the number of registers that can be used. This abstraction is valid where the size of the problem is small enough to fit in the main memory of a computer, or the integers used in the computation are small enough to fit in one computer word. All computations take place in the first register r_0 , called accumulator, which can hold an arbitrary integer. The program for a RAM is not stored in memory and cannot modify itself. The programs are just a sequence of labeled instructions. The instructions can be arithmetic instructions, input/output instructions, indirect addressing and branching instructions.



2. Operand Types in RAM

An operand of RAM instructions can be one of the following:

1. $= i$, indicating the integer itself. E.g., LOAD = 3, ADD = 3, MULT = 3
2. A non-negative integer i , indicating the contents of register. E.g., LOAD 1, STORE 1, SUB 1
3. $* i$, indicating indirect addressing. E.g., LOAD * 2, ADD * 1

3. Logarithmic Costs of RAM Instructions

Instruction	Cost
1. LOAD a	$t(a)$
2. STORE i	$l(c(0)) + l(i)$
STORE $*i$	$l(c(0)) + l(i) + l(c(i))$
3. ADD a	$l(c(0)) + t(a)$
4. SUB a	$l(c(0)) + t(a)$
5. MULT a	$l(c(0)) + t(a)$
6. DIV a	$l(c(0)) + t(a)$
7. READ i	$l(\text{input}) + l(i)$
READ $*i$	$l(\text{input}) + l(i) + l(c(i))$
8. WRITE a	$t(a)$
9. JUMP b	1
10. JGTZ b	$l(c(0))$
11. JZERO b	$l(c(0))$
12. HALT	1

4. Random Access Stored Program Model

The random access stored program model is similar to RAM with the exception that the program is in memory and can modify itself. But indirect addressing is not permitted. Each RASP instruction occupies two consecutive registers, the first register holds the encoding of the operation code, and the second register holds the address.

5. Operation Codes of RASP Instructions

Instruction	Encoding	Instruction	Encoding
LOAD i	1	DIV i	10
LOAD $=i$	2	DIV $=i$	11
STORE i	3	READ i	12
ADD i	4	WRITE i	13
ADD $=i$	5	WRITE $=i$	14
SUB i	6	JUMP i	15
SUB $=i$	7	JGTZ i	16
MULT i	8	JZERO i	17
MULT $=i$	9	HALT	18

6. Costs of RASP Instructions (SUB *i)

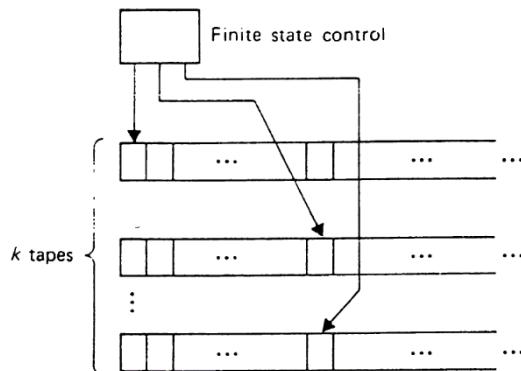
RASP register	Instruction	Cost
j	STORE 1	$l(j) + l(1) + l(c(0))$
$j+2$	LOAD $r+i$	$l(j+2) + l(r+i) + l(c(i))$
$j+4$	ADD $=r$	$l(j+4) + l(c(i)) + l(r)$
$j+6$	STORE $j+11$	$l(j+6) + l(j+11) + l(c(i) + r)$
$j+8$	LOAD 1	$l(j+8) + l(1) + l(c(0))$
$j+10$	SUB -	$l(j+10) + l(c(i) + r) + l(c(0)) + l(c(c(i)))$

7. Difference between RAM and RASP

RAM	RASP
The program is not stored in memory and instructions are not permitted to modify themselves.	The program is stored in memory and can modify itself.
Indirect addressing is permitted.	Indirect addressing is not permitted.
The program is merely a sequence of labeled instructions.	Each RASP instruction occupies two consecutive memory registers, one for an encoding of operation code and one for the address.

8. Multi-tape Turing Machine

A multi-tape Turing machine consists of some number k of tuples, which are infinite to the right. Each tape is marked off into cells, each of which holds one of a finite number of tape symbols. One cell on each tape is scanned by a tape head, which can read and write. The operation of the Turing machine is determined by a primitive program called a finite control. The finite control is always in one of a finite number of states, which can be regarded as positions of the program.



Chapter 2

Design of Efficient Algorithms

ရွှေအခန်းမှာတော့ ကျွန်တော်တို့အနေနဲ့ algorithm တွေရဲ့ performance ကို တန်းတူရည်တူနှင့်ယူဉ်ပြီး analysis လုပ်နိုင်ဖို့အတွက် computational model တွေကို လေ့လာခဲ့ကြပါတယ်။ Algorithm တွေရဲ့ performance ဟာ သူကိုတင်ပြီး run တဲ့ computational model သဘောသဘာဝအပေါ်လဲ မူတည်သလို algorithm ကိုယ်နှိုက်ရဲ့ design နဲ့လည်ပါတယ်။ Efficient လဲဖြစ်၊ ရိုးလဲရိုးရှင်းတဲ့ algorithm တွေဖန်တီးရာမှာ အဓိကလိုအပ်တဲ့ programming technique တွေကို ဒီအခန်းမှာ လေ့လာသွားကြပါမယ်။ ဒီထဲမှာ လူသုံးများတဲ့ နာမည်ကြီး divide and conquer approach အကြောင်း balancing အကြောင်းတွေအပြင် dynamic programming concept တွေပါ ပါဝင်မှာဖြစ်ပါတယ်။

2.1 Divide and Conquer Approach

Problem အကြိုးကြီးတွေကို တစ်ကြိမ်တည်းအပြီး ဖြေရှင်းတာထက် problem အသေးစားလေးတွေဖြစ်အောင်ခဲ့ပြီး ဖြေရှင်းတာက ပိုလွယ်ပါတယ်။ ဒီနည်းအရ ခဲ့ခြမ်းလိုက်တဲ့ subproblem လေးတွေ တစ်ခုစီကို သီးခြားဆီ ဖြေရှင်းပြီး ရလာတဲ့ solution တွေကို တစ်ခုတည်းဖြစ်အောင် ပြန်ပေါင်းလိုက်တာပါ။ ဒီ approach ကို recursion concept နဲ့ ပူးတွဲအသုံးပြုလိုက်တဲ့အခါ ပိုမိုကျစ်လျှစ် ရိုးရှင်းပြီး performance ကောင်းတဲ့ algorithm တွေ ရရှိလာဖော်ပါတယ်။

ဥပမာအနေနဲ့ set တစ်ခုထဲက အကြိုးဆုံးနဲ့ အငယ်ဆုံး element တန်ဖိုးတွေရှာတာကို စဉ်းစားကြည့်ပါ။ စဉ်းစား တွက်ချက်ရတာ ပိုလွယ်သွားအောင် element အရေအတွက်သည် 2 ပါဝါတန်ဖိုး (2, 4, 8, 16,...) ဖြစ်တယ်လို့ ယူဆပါ။ ဒီလို့ maximum minimum ရှာရာမှာ လေ့လာတဲ့ ရိုးရှင်းနည်းဖြစ်တဲ့ ပါဝင် element တစ်ခုစီကို နှင့်ယူဉ်တဲ့နည်းနဲ့တွက်လိုလဲ ရပါတယ်။ Figure 2.1 က algorithm ကိုကြည့်ပါ။

```

begin
    MAX ← any element in S:
    for all other elements x in S do
        if x > MAX then MAX ← x
    end

```

Figure 2.1 Max, min တန်ဖိုးရှာဖို့ traditional method

ဒီနည်းနဲ့တွက်ရင် input n ခုအတွက် comparison အကြိမ်ရောစွာပေါင်း $2n - 3$ လိုအပ်ပါတယ်။ ဒါပေမဲ့ ဒီပုံစွာကို နောက်တစ်နည်းနဲ့လဲ တွက်လိုပါတယ်။ ပေးထားတဲ့ အစုကို နှစ်ပိုင်းစီ ပိုင်းပိုင်းသွားပြီး တစ်ပိုင်းစီအတွက် max, min တန်ဖိုးတွေရှာတဲ့နည်းကို Figure 2.2 က algorithm မှာ ကြည့်ပါ။

```

procedure MAXMIN(S):
1. if  $\|S\| = 2$  then
    begin
2.     let  $S = \{a, b\}$ :
3.     return (MAX(a, b), MIN(a, b))
    end
else

```

```

begin
4.      divide  $S$  into two subsets  $S_1$  and  $S_2$ , each with half the elements;
5.      ( $\max_1, \min_1$ )  $\leftarrow \text{MAXMIN}(S_1)$ ;
6.      ( $\max_2, \min_2$ )  $\leftarrow \text{MAXMIN}(S_2)$ ;
7.      return ( $\text{MAX}(\max_1, \max_2), \text{MIN}(\min_1, \min_2)$ )
end

```

Figure 2.2 Max, min ရှာဖို့ divide and conquer method

Divide and conquer method အရ ဖြေရှင်းရင်တော့ input အရေအတွက် n ခုအတွက် comparison အကြိမ်ရေစွဲပေါင်း $\frac{3}{2}n - 2$ ပဲ လိုတာကို တွေ့ရပါတယ်။ (ဒီလို comparison အကြိမ်အရေ အတွက် တွက်ချက်ထားတာတွေကို reference textbook ထဲမှာ ရှုပ်ထွေးပွဲလိုစာ တွက်ပြထားပါတယ်။ ဒါကြောင့် စိတ်ဝင်စားမှပဲ သွားဖတ်ကြည့်ပါ။ မဟုတ်ရင် ဒီတိုင်းပမှတ်ထားပါ။) ဒါကြောင့် comparison အကြိမ်အရေအတွက်အရကြည့်ရင် divide and conquer method က အနည်းငယ်ပိုမြန် တာကို တွေ့နိုင်ပါတယ်။ Max, min ကို divide and conquer method နဲ့ရှာထားတဲ့ ဥပမာကို Example 2.1 မှာကြည့်ပါ။

Example 2.1 Find the maximum and minimum of elements of a set S with elements 5, 7, 6, 4, 3, 5, 1, 2 using divide and conquer approach.

Solution:

$$S = \{5, 7, 6, 4, 3, 5, 1, 2\},$$

$$\text{MAXMIN}(S) \Rightarrow S_1 = \{5, 7, 6, 4\}, S_2 = \{3, 5, 1, 2\}$$

$$(\max_1, \min_1) \leftarrow \text{MAXMIN}(S_1), \quad (\max_2, \min_2) \leftarrow \text{MAXMIN}(S_2)$$

$$\text{MAXMIN}(S_1) \Rightarrow S_1 = \{5, 7\}, S_2 = \{6, 4\}$$

$$(\max_1, \min_1) \leftarrow \text{MAXMIN}(S_1), \quad (\max_2, \min_2) \leftarrow \text{MAXMIN}(S_2)$$

$$\text{MAXMIN}(S_1 = \{5, 7\}) \Rightarrow \text{return } (\text{MAX}(5, 7), \text{MIN}(5, 7)) \Rightarrow \text{return } (7, 5)$$

$$\text{MAXMIN}(S_2 = \{6, 4\}) \Rightarrow \text{return } (\text{MAX}(6, 4), \text{MIN}(6, 4)) \Rightarrow \text{return } (6, 4)$$

$$\text{return } (\text{MAX}(7, 6), \text{MIN}(5, 4)) \Rightarrow \text{return } (7, 4)$$

$$\text{MAXMIN}(S_2) \Rightarrow S_1 = \{3, 5\}, S_2 = \{1, 2\}$$

$$(\max_1, \min_1) \leftarrow \text{MAXMIN}(S_1), \quad (\max_2, \min_2) \leftarrow \text{MAXMIN}(S_2)$$

$$\text{MAXMIN}(S_1 = \{3, 5\}) \Rightarrow \text{return } (\text{MAX}(3, 5), \text{MIN}(3, 5)) \Rightarrow \text{return } (5, 3)$$

$$\text{MAXMIN}(S_2 = \{1, 2\}) \Rightarrow \text{return } (\text{MAX}(1, 2), \text{MIN}(1, 2)) \Rightarrow \text{return } (2, 1)$$

$$\text{return } (\text{MAX}(5, 2), \text{MIN}(3, 1)) \Rightarrow \text{return } (5, 1)$$

$$\text{return } (\text{MAX}(7, 5), \text{MIN}(4, 1)) \Rightarrow \text{return } (7, 1)$$

$$\therefore \max = 7, \min = 1$$

Divide and conquer နည်းရဲ့ အားသာချက်ကို ဖော်ပြနိုင်တဲ့ နောက် ဥပမာတစ်ခုရှိပါတယ်။ ဒါက တော့ sorting လုပ်ခြင်းပဲဖြစ်ပါတယ်။ Sorting algorithm များစွာထဲကမှ Mergesort algorithm မှာ ဒီနည်းကို တွင်တွင်ကျယ်ကျယ် အသုံးပြုထားပါတယ်။ Divide and conquer သုံးတာချင်းအတူတူ subproblem ခဲ့တဲ့အခါ subproblem အရွယ်အစား ညီတူမျှတဲ့ ခဲ့တာ တစ်နည်း balancing ညီထားတာက efficient ပိုဖြစ်တယ်ဆိုတာကို ဒီ algorithm က ဖော်ပြပါတယ်။ ဒီ algorithm အရ အစုတစ်ခုကို ပါဝင်

```

procedure SORT( $i, j$ ):
if  $i = j$  then return  $x_i$ 
else
    begin
         $m \leftarrow (i + j - 1)/2;$ 
        return MERGE(SORT( $i, m$ ), SORT( $m + 1, j$ ))
    end

```

Figure 2.3 Mergesort algorithm

element တစ်ခု တည်းရှုတဲ့အထိ နှစ်ခြမ်းပိုင်းပိုင်းပြီး တစ်ပိုင်းစီကို sorting စီတာဖြစ်ပါတယ်။ Sorting စီထားတဲ့ အစိတ်အပိုင်း တစ်ခုစီကို ပြန်ပြီး merge လုပ်တဲ့အခါ လိုချင်တဲ့ sorted set ကို ရရှိမှာဖြစ်ပါတယ်။ Algorithm ကို Figure 2.3 မှာ ကြည့်ပါ။ Time complexity ကတေသာ $O(n \log n)$ ဖြစ်ပါတယ်။

Example 2.2. Sort the sequence 5, 8, 3, 9, 6, 4, 1, 7 by Mergesort.

Solution:

5, 8, 3, 9, 6, 4, 1, 7

```

SORT(1, 8){
 $1 \neq 8, mid = \lfloor \frac{(8+1)}{2} \rfloor = 4$ 
    SORT(1, 4){
         $1 \neq 4, mid = \lfloor \frac{4+1}{2} \rfloor = 2$ 
            SORT(1, 2){
                 $1 \neq 2, mid = \lfloor \frac{(2+1)}{2} \rfloor = 1$ 
                    SORT(1, 1){
                         $1 = 1, return 5;$ 
                    }
                    SORT(2, 2){
                         $2 = 2, return 8;$ 
                    }
                    MERGE(1, 2){
                         $return (5, 8);$ 
                    }
                }
                SORT(3, 4){
                     $3 \neq 4, mid = \lfloor \frac{(3+4)}{2} \rfloor = 3$ 
                        SORT(3, 3){
                             $3 = 3, return 3;$ 
                        }
                        SORT(4, 4){
                             $4 = 4, return 9;$ 
                        }
                        MERGE(3, 4){
                             $return (3, 9);$ 
                        }
                    }
                    MERGE(1, 4){
                         $return (3, 5, 8, 9);$ 
                    }
    }

```

```

SORT(5, 8){
     $5 \neq 8, mid = \lfloor \frac{5+8}{2} \rfloor = 6$ 
    SORT(5, 6){
         $5 \neq 6, mid = \lfloor \frac{(5+6)}{2} \rfloor = 5$ 
        SORT(5, 5){
             $5 = 5, return 6;$ 
        }
        SORT(6, 6){
             $6 = 6, return 4;$ 
        }
        MERGE(5, 6){
             $return (4, 6);$ 
        }
    }
    SORT(7, 8){
         $7 \neq 8, mid = \lfloor \frac{(7+8)}{2} \rfloor = 7$ 
        SORT(7, 7){
             $7 = 7, return 1;$ 
        }
        SORT(8, 8){
             $8 = 8, return 7;$ 
        }
        MERGE(8, 7){
             $return (1, 7);$ 
        }
    }
    MERGE(5, 8){
         $return (1, 4, 6, 7);$ 
    }
}
MERGE(1, 8){
     $return (1, 3, 4, 5, 6, 7, 8, 9);$ 
}
}

```

2.2 Dynamic Programming

Dynamic programming ဆိုတာဟာ divide and conquer method တစ်မျိုးဖြစ်ပါတယ်။ Problem တစ်ခုကို subproblem တွေခဲ့ပြီး ဖြေရှင်းတာချင်းတူပေမဲ့ dynamic programming ရဲ့ အားသာ ချက်က ဖြေရှင်းပြီးသား subproblem result တွေကို table တစ်ခုထဲမှာ စုသိမ်းထားတာဖြစ်တဲ့အတွက် ဖြေရှင်းပြီးသား subproblem တွေကို ထပ်ခါထပ်ခါ ဖြေရှင်းရတာမျိုး မရှိတော့ပါဘူး။ ဉာပမာတစ်ခုအနေနဲ့

```

begin
1.   for  $i \leftarrow 1$  until  $n$  do  $m_{ii} \leftarrow 0;$ 
2.   for  $l \leftarrow 1$  until  $n - 1$  do
3.       for  $i \leftarrow 1$  until  $n - l$  do
                begin
4.                     $j \leftarrow i + l;$ 
5.                     $m_{ij} \leftarrow \text{MIN}_{i \leq k < j} (m_{ik} + m_{k+1,j} + r_{i-1} * r_k * r_j)$ 
                end;
6.       write  $m_{1n}$ 
end

```

Figure 2.4 Matrix multiplication အတွက် Dynamic Programming algorithm

matrix တွေပေးထားပြီး operation အရေအတွက် အနည်းဆုံးနဲ့ မြောက်လဒ် အစိအစဉ်ကို ရှာရာမှာ ဒီ dynamic programming method ကို သုံးနိုင်ပါတယ်။ ဒီလို matrix multiplication အတွက် လိုအပ်တဲ့ algorithm ကို Figure 2.4 မှာ ဖြည့်နိုင်ပါတယ်။

Matrix လေးချပေးထားမယ်ဆိုရင် ဒီ dynamic programming အရ matrix dimension value တွေကို $r_0, r_1, r_2, \dots, r_4$ ဆိုပြီး သတ်မှတ်ပေးရပါမယ်။ ပြီးတဲ့ အခါ $m_{11}, m_{22}, m_{33}, m_{44}$ ကို သူညာသတ်မှတ်ပြီး $m_{ij} \leftarrow \min_{i \leq k < j} (m_{ik} + m_{k+1,j} + r_{i-1} * r_k * r_j)$ formula သုံးရပါမယ်။ $l = 1 \text{ to } l = 3$ အထိ တွက်ပေးရပါမယ်။ Minimum operator ထဲမှာ m တန်ဖိုးတစ်ခုထက်မက ရှိနေရင် နောက်ဆုံးရလဒ် တန်ဖိုးအနည်းဆုံးကိုယူရပါမယ်။ နောက်ဆုံး m_{14} ရတဲ့ အခါ order of execution ကိုရှာဖို့ m_{14} ကို နောက်ပြန်ခဲ့ပြီး ယူရပါမယ်။

Example 2.3. Find the minimal number of operations in multiplying the matrices –

$$\begin{array}{c} M1 \\ [10 \times 20] \end{array} \times \begin{array}{c} M2 \\ [20 \times 50] \end{array} \times \begin{array}{c} M3 \\ [50 \times 1] \end{array} \times \begin{array}{c} M4 \\ [1 \times 100] \end{array}$$

Also find the order of execution to multiply.

Solution:

$$\begin{aligned} n &= 4, r_0 = 10, r_1 = 20, r_2 = 50, r_3 = 1, r_4 = 100 \\ m_{11} &= 0, m_{22} = 0, m_{33} = 0, m_{44} = 0 \\ m_{ij} &= \min_{i \leq k < j} (m_{ik} + m_{k+1,j} + r_{i-1} * r_k * r_j) \end{aligned}$$

For $l = 1, i = 1, j = 2, k = 1$,

$$m_{12} = \min(m_{11} + m_{22} + r_1 * r_2 * r_3) = \min(0 + 0 + 10 * 20 * 50) = 10000$$

For $l = 1, i = 2, j = 3, k = 2$,

$$m_{23} = \min(m_{22} + m_{33} + r_1 * r_2 * r_3) = \min(0 + 0 + 20 * 50 * 1) = 1000$$

For $l = 1, i = 3, j = 4, k = 3$,

$$m_{34} = \min(m_{33} + m_{44} + r_2 * r_3 * r_4) = \min(0 + 0 + 50 * 1 * 100) = 5000$$

For $l = 2, i = 1, j = 3, k = 1, 2$

$$\begin{aligned} m_{13} &= \min[(m_{11} + m_{23} + r_0 * r_1 * r_3), (m_{12} + m_{33} + r_0 * r_2 * r_3)] \\ &= \min[(0 + 1000 + 10 * 20 * 1), (10000 + 0 + 10 * 50 * 1)] \\ &= \min(1200, 10500) = 1200 \end{aligned}$$

For $l = 2, i = 2, j = 4, k = 3, 4$

$$\begin{aligned} m_{24} &= \min[(m_{22} + m_{34} + r_1 * r_2 * r_4), (m_{23} + m_{44} + r_1 * r_3 * r_4)] \\ &= \min[(0 + 5000 + 20 * 50 * 1), (1000 + 0 + 20 * 1 * 100)] \\ &= \min(51000, 3000) = 3000 \end{aligned}$$

For $l = 3, i = 1, j = 4, k = 1, 2, 3$

$$\begin{aligned} m_{14} &= \min[(m_{11} + m_{24} + r_0 * r_1 * r_4), (m_{12} + m_{34} + r_0 * r_2 * r_4), (m_{13} + m_{44} + r_0 * r_3 * r_4)] \\ &= \min[(0 + 3000 + 10 * 20 * 100), (10000 + 5000 + 10 * 50 * 100), (1200 + 0 + 10 * 1 * 100)] \\ &= \min(23000, 65000, 22000) \\ &= 22000 \end{aligned}$$

The order of execution for the minimal number of operations is

$$\begin{aligned} M_{14} &= m_{13} + m_{44} = (m_{11} + m_{23}) + m_{44} = (m_{11} + (m_{22} + m_{33})) + m_{44} \\ &\therefore (M_1 * (M_2 * M_3)) * M_4 \end{aligned}$$

Exercise 2.1. Dynamic Programming

Find the minimal number of operations in multiplying the matrices –

$$\begin{matrix} M_1 & \times & M_2 & \times & M_3 & \times & M_4 \\ [20 \times 10] & & [10 \times 30] & & [30 \times 100] & & [100 \times 1] \end{matrix}$$

Also find the order of execution to multiply.

Conceptual Notes

1. Divide and Conquer Approach

A common approach to solving a problem which involves partitioning the problem in smaller parts, find the solution for the parts and then combine the solutions for the parts into a solution for the whole. This approach, especially when used recursively, often yields efficient solutions to problems in which the sub-problems are smaller versions of the original problem.

2. MAXMIN Procedure (traditional)

```

begin
    MAX  $\leftarrow$  any element in S;
    for all other elements x in S do
        if x > MAX then MAX  $\leftarrow$  x
    end

```

3. MAXMIN Procedure (recursive)

```

procedure MAXMIN(S):
    if  $\|S\| = 2$  then
        begin
            let S = {a, b};
            return (MAX(a, b), MIN(a, b))
        end
    else
        begin
            divide S into two subsets  $S_1$  and  $S_2$ , each with half the elements:
            (max1, min1)  $\leftarrow$  MAXMIN( $S_1$ );
            (max2, min2)  $\leftarrow$  MAXMIN( $S_2$ );
            return (MAX(max1, max2), MIN(min1, min2))
        end

```

4. Mergesort Procedure

```

procedure SORT(i, j):
    if i = j then return  $x_i$ 
    else
        begin
            m  $\leftarrow$   $(i + j - 1)/2$ ;
            return MERGE(SORT(i, m), SORT(m + 1, j))
        end

```

5. Dynamic Programming

Dynamic programming calculates the solutions to all sub-problems. The computation proceeds from the small sub-problems to the larger sub-problems, storing answers in a table. The advantage of the method lies in the fact that once a sub-problem is solved, the answer is stored and never recalculated.

6. Dynamic Programming Procedure

```
begin
    for i ← 1 until n do  $m_{ii} \leftarrow 0$ ;
    for l ← 1 until n - 1 do
        for i ← 1 until n - l do
            begin
                j ← i + l;
                 $m_{ij} \leftarrow \text{MIN}_{i \leq k \leq j} (m_{ik} + m_{k+1,j} + r_{i-1} * r_k * r_j)$ 
            end;
        write  $m_{in}$ 
    end
```

Chapter 3

Sorting and Order Statistics

Sorting နဲ့ order statistics ဆိုတာဟာ လေ့လာလိုလဲမကုန်၊ စိတ်ဝင်စားဖို့လဲ ကောင်းတဲ့အပြင် လက်တွေ့ နည်းပညာနယ်ပယ်ထဲမှာလဲ အင်မတန်အရေးပါတဲ့ သဘောတရားနှစ်ခုဖြစ်ပါတယ်။ ဒီသဘောတရားနှစ်ခုကို အထူးသဖြင့် database indexing, data visualization, ranking system တွေလိုမျိုး ကြိုးမား မြောက်မြားတဲ့ data တွေကို ကိုင်တွယ်ရာမှာ တွင်တွင်ကျယ်ကျယ် အသုံးပြုပါတယ်။ ဒီအခန်းမှာတော့ sorting အမျိုးအစားများစွာတဲ့ radix sort, heapsort, quicksort သုံးမျိုးအကြောင်းရယ်၊ နောက်ဆုံး အပိုင်းမှာ order statistics အကြောင်းရယ် လေ့လာသွားကြပါမယ်။ Radix sort ဆိုတာသည် sorting စီမယ့် data element တစ်ခုစီရဲ့ ဖွဲ့စည်းတည်ဆောက်ပုံကို အသုံးပြုပြီး sorting စီတာဖြစ်ပါတယ်။ ကျန်တဲ့နှစ်ခုက တော့ comparison တွေလုပ်ပြီးပဲ sorting စီတာ ဖြစ်ပါတယ်။

3.1 Radix Sort

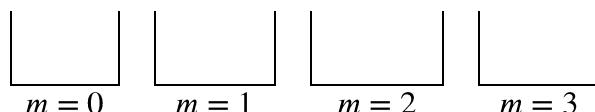
Sorting method အတော်များများဟာ data value တွေကို comparison လုပ်ပြီး sort လုပ်ကြပါတယ်။ Radix sort ကတော့ comparison မလုပ်ဘဲ data value ထဲပါတဲ့ data item တစ်ခုစီကို မျိုးတူ bucket တွေထဲစုသိမ်းရင်း sorting စီတာဖြစ်ပါတယ်။ Radix sort algorithm အရ sorting စီမယ့် digit တွေကို digit တန်ဖိုးတစ်ခုစီအတွက် bucket တစ်ခုစီ ဖန်တီးပောမှာဖြစ်ပြီး တန်ဖိုးတူ digit တွေကို အဲဒီ bucket တွေထဲ ထည့်ပေးရမှာဖြစ်ပါတယ်။ ထည့်လိုကုန်ပြီဆိုရင် bucket အားလုံးကို အစဉ်လိုက် ဆက်ပေးရမှာ ဖြစ်ပါတယ်။

Example 3.1. Sort the sequence 2, 0, 3, 1, 2, 1 using radix sort.

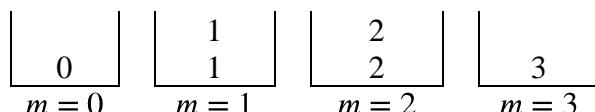
Solution:

$$m = 4, n = 6$$

Step 1:



Step 2:



Step 3: 0, 1, 1, 2, 2, 3

Bucket sort လိုလဲခေါ်တဲ့ radix sort ဟာ digit တစ်လုံးတည်း ပါဝင်တဲ့ element တွေကိုပဲ ကိုင်တွယ်နိုင်ပါတယ်။ ဒီလိုမဟုတ်ဘဲ digit တစ်လုံး၊ character တစ်လုံးထက်မက ပါဝင်တဲ့ data တွေကို sort လုပ်မယ်ဆိုရင် radix sort နဲ့ ဆင်တူတဲ့ lexicographic sort ကို သုံးနိုင်ပါတယ်။ Lexicographic sort မှာလဲ ပါဝင် data value တွေထဲက digit or character အရေအတွက်ပေါ်မှုတည်ပြီး fixed length algorithm နဲ့ varying length algorithm ဆိုပြီး နှစ်ခုရှိပါတယ်။

အလျားတူ string တန်ဖိုးတွေ sort ချင်ရင် fixed length algorithm ကိုသုံးရမှာဖြစ်ပြီး အလျားမတူတဲ့ string တွေ sorting စီချင်ရင်တော့ varying length algorithm အတိုင်း စီရမှာဖြစ်ပါတယ်။

```

begin
    place  $A_1, A_2, \dots, A_n$  in QUEUE;
    for  $j \leftarrow k$  step  $-1$  until  $1$  do
        begin
            for  $l \leftarrow 0$  until  $m - 1$  do make  $Q[l]$  empty;
            while QUEUE not empty do
                begin
                    let  $A_i$  be the first element in QUEUE;
                    move  $A_i$  from QUEUE to bucket  $Q[a_{ij}]$ 
                end;
            for  $l \leftarrow 0$  until  $m - 1$  do
                concatenate contents of  $Q[l]$  to the end of QUEUE
        end
    end

```

Figure 3.1 Lexicographic Sort (Fixed Length)

Fixed-length algorithm ကို Figure 3.1 မှာ တွေ့နှင့်ပါတယ်။ ဒီ algorithm အရ string ထဲ ပါသော character တွေကို QUEUE ထဲအရင်ထည့်ပါတယ်။ ပြီးတဲ့နောက် character တစ်လုံးစီအတွက် bucket တစ်လုံးစီ အစဉ်အတိုင်း အရင်လုပ်ပြီး string တစ်ခုစို့ နောက်ကနေ စပြီး သက်ဆိုင် ရာ bucket ထဲ ထည့်ရပါတယ်။ Bucket တွေထဲ တစ်ခါထည့်ပြီးတိုင်း concatenate တစ်ခါပြန်လုပ်ပါတယ်။ ဒီလို့ bucket ထဲထည့်ပြီး sorting စီရတဲ့ အကြိမ်အရေအတွက်သည် string တစ်ခုစီမှာပါတဲ့ character အရေ အတွက်နဲ့ ညီပါတယ်။ Time complexity အနေနဲ့ $O((m + n)k)$ ရှိပါတယ်။

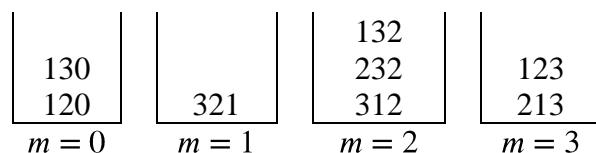
Example 3.2. Sort the sequence 312, 213, 123, 120, 232, 321, 132, 130 using lexicographic sort.

Solution:

$$n = 8, k = 3, m = 4$$

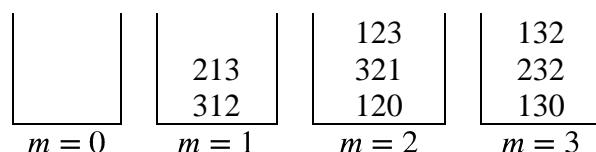
$$\text{QUEUE} = \{312, 213, 123, 120, 232, 321, 132, 130\}$$

For $j = 3$, move the elements from QUEUE to buckets.



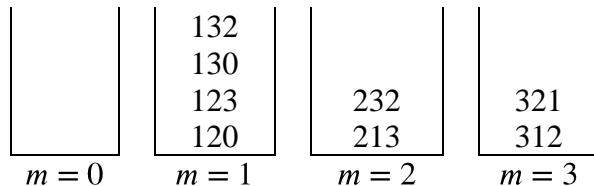
After concatenation, $\text{QUEUE} = \{120, 130, 321, 312, 232, 132, 213, 123\}$

For $j = 2$,



After concatenation, $\text{QUEUE} = \{312, 213, 120, 321, 123, 130, 232, 132\}$

For $j = 1$,



After concatenation, $QUEUE = \{120, 123, 130, 132, 213, 232, 312, 321\}$

Length မတူတဲ့ string တွေကိုတော့ varying length algorithm အရ အလျားအရှည်ဆုံး string က နေစပြီး အတိုဆုံး string အထိ သူနေရာအလိုက် bucket ထဲထည့်သွားတာဖြစ်ပါတယ်။ အရင်ဆုံး string တစ်ခုစိတိ index နဲ့ character တဲ့ထားတဲ့ ordered pair လေးတွေ ချရေးပါတယ်။ ပြီးရင် အဲဒီအတွဲလေး တွေကို index ပေါ်စဉ်ကြီးလိုက် စီရပါတယ်။ ရလာတဲ့ ordered pair လေးတွေမှာ index တန်ဖိုးတူရာ character တွေကို NONEMPTY ဆိုတဲ့ array ထဲ ထည့်ရပါမယ်။ ပြီးတဲ့အခါ LENGTH ဆိုတဲ့ array ထဲမှာ string length အလိုက် ထည့်ပေးပြီး length အရှည်ဆုံးကနေ အတိုဆုံးထိ fixed length algorithm လို ဆက်လုပ်သွားတာဖြစ်ပါတယ်။ Varying length algorithm ကို Figure 3.2 မှာကြည့်ပါ။ Time complexity အနေနဲ့ $O(l_{total} + m)$ ရှုပါတယ်။

```

begin
1.    make QUEUE empty;
2.    for  $j \leftarrow 0$  until  $m - 1$  do make  $Q[j]$  empty;
3.    for  $l \leftarrow l_{max}$  step  $-1$  until  $1$  do
        begin
            4.        concatenate LENGTH[ $l$ ] to the beginning of
                    QUEUE;
            5.        while QUEUE not empty do
                begin
                    6.            let  $A_i$  be the first string on QUEUE;
                    7.            move  $A_i$  from QUEUE to bucket  $Q[a_i]$ 
                end;
            8.            for each  $j$  on NONEMPTY[ $l$ ] do
                begin
                    9.                concatenate  $Q[j]$  to the end of QUEUE;
                    10.               make  $Q[j]$  empty
                end
            end
        end
    end
end

```

Figure 3.2 Lexicographic Sort (Varying Length)

Example 3.3. Sort the strings $a, bc, aab, baca, cbc, cc$ using lexicographic sort.

Solution:

The pairs are

- (1, a),
- (1, b), (2, c),
- (1, a), (2, a), (3, b),
- (1, b), (2, a), (3, c), (4, a),
- (1, c), (2, b), (3, c)
- (1, c), (2, c)

Sorted pairs are

(1, a), (1, a), (1, b), (1, b), (1, c), (1, c),

$(2, a), (2, a), (2, b), (2, c), (2, c),$
 $(3, b), (3, b), (3, c),$
 $(4, a)$

$NONEMPTY[1] = a, b, c$

$NONEMPTY[2] = a, b, c$

$NONEMPTY[3] = b, c$

$NONEMPTY[4] = a$

$$l_1 = 1, l_2 = 2, l_3 = 3, l_4 = 1$$

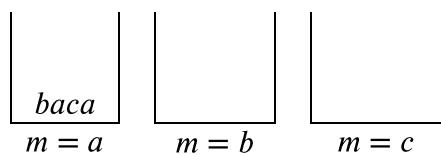
$LENGTH[1] = a$

$LENGTH[2] = bc, cc$

$LENGTH[3] = aab, cbc$

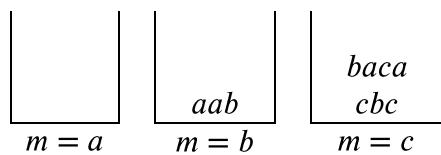
$LENGTH[4] = baca$

For $QUEUE = \{baca\}, l = 4$



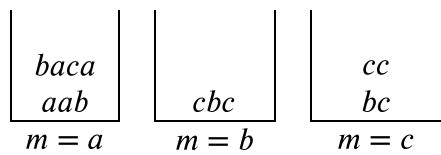
After concatenation, $QUEUE = \{baca\}$

For $QUEUE = \{aab, cbc, baca\}, l = 3$



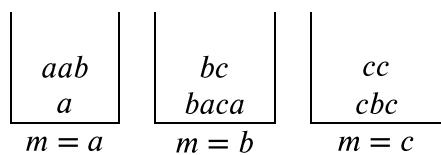
After concatenation, $QUEUE = \{aab, cbc, baca\}$

For $QUEUE = \{bc, cc, aab, cbc, baca\}, l = 2$



After concatenation, $QUEUE = \{aab, baca, cbc, bc, cc\}$

For $QUEUE = \{a, aab, baca, cbc, bc, cc\}, l = 1$



After concatenation, $QUEUE = \{a, aab, baca, bc, cbc, cc\}$

3.2 Heapsort

Parent ဟာ child node နှစ်ခုထက် ကြီးရုံးနဲ့ အသုံးပြုလိုရတဲ့ binary tree အမျိုးအစားကို heap လို ခေါ်ပါတယ်။ Heap တစ်ခုသည် heap characteristics အရ parent အားလုံးဟာ child node တွေထက် ကြီးနေဖို့လိုပါတယ်။

Heap တစ်ခု တည်ဆောက်ရင် node တစ်လုံးချင်းထည့်ပြီး ဆောက်လို့ရသလို unordered heap အရင်ဆောက်ပြီးမှ trickle up, trickle down လုပ်လိုလဲရနိုင်ပါတယ်။ Heap ထဲ ထည့်လိုက်တဲ့ node အသစ်တွေကို heap condition အရ နေရာမှန်ရောက်အောင် နေရာအဆင့်ဆင့်လဲတာကို trickle လုပ်တယ်လို့ခေါ်ပါတယ်။ Heap အသစ်ဆောက်ရာမှာ node တစ်လုံးချင်းထည့်ဆောက်ရင် အသစ်ထည့်လိုက်တဲ့ node တွေကို heap ရဲ့ နောက်ဆုံးနေရာမှာတပ်၊ trickle up လုပ် (တစ်နည်း သူ့နေရာမှန်ရောက်အောင် အပေါ်က node တွေနဲ့ နေရာ အဆင့်ဆင့်လဲ) ပေးရပါမယ်။

Heap တွေသုံးပြီး sorting ဖိတာကို heapsort လို့ခေါ်ပါတယ်။ Time complexity အနေနဲ့ $O(n \log n)$ ရှိပါတယ်။ Heapsort သုံးတဲ့အခါ -ပေးထားတဲ့ element တွေကို heap အရင်ဆောက်၊ Root ကိုဖယ်၊ array နောက်ဆုံးကိုပို့ ဖယ်တဲ့နေရာကို last node ထည့် trickle down လုပ်ရပါမယ်။ Heapsort တစ်ခုတည်းမှာ procedure အခဲနှစ်ခု ပါဝင်ပါတယ်။ ပုံစွာက ပေးထားတဲ့ တန်ဖိုးတွေကို heap ဆောက်ရာမှာ BUILDHEAP procedure ကိုသုံးပြီး subtree တွေ ပြန်ဆောက်ရာမှာတော့ HEAPIFY procedure ကိုသုံးပါတယ်။ Figure 3.3 မှာ ကြည့်နိုင်ပါတယ်။

```

begin
    BUILDHEAP;
    for  $i \leftarrow n$  step  $-1$  until  $2$  do
        begin
            interchange  $A[1]$  and  $A[i]$ ;
            HEAPIFY( $1, i - 1$ )
        end
    end □
procedure HEAPIFY( $i, j$ ):
if  $i$  is not a leaf and if a son of  $i$  contains a larger element than  $i$ 
    does then
        begin
            let  $k$  be a son of  $i$  with the largest element:
            interchange  $A[i]$  and  $A[k]$ ;
            HEAPIFY( $k, j$ )
        end
procedure BUILDHEAP:
for  $i \leftarrow n$  step  $-1$  until  $1$  do HEAPIFY( $i, n$ )

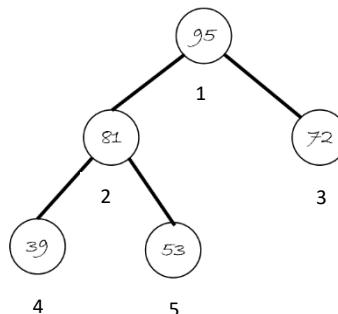
```

Figure 3.3 Heapsort Algorithm

Example 3.3. Using heap sort, arrange the values 95, 81, 72, 39, 53 in an ascending order.

Solution:

Heap construction using *BUILDHEAP* procedure,

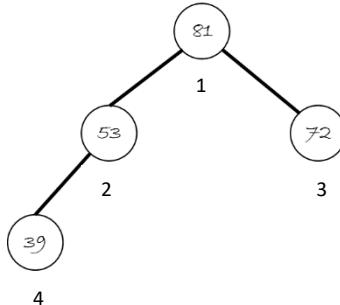


Heapsort using *HEAPIFY* procedure,

Values = {95, 81, 72, 39, 53}

For $i = 5$, interchange $A[1]$ and $A[5]$

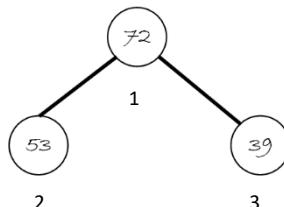
HEAPIFY(1, 4)



Values = {81, 53, 72, 39, 95}

For $i = 4$, interchange $A[1]$ and $A[4]$

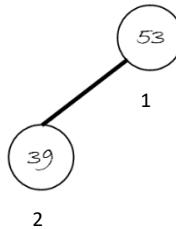
HEAPIFY(1, 3)



Values = {72, 53, 39, 81, 95}

For $i = 3$, interchange $A[1]$ and $A[3]$

HEAPIFY(1, 2)



Values = {53, 39, 72, 81, 95}

For $i = 2$, interchange $A[1]$ and $A[2]$

HEAPIFY(1, 1)

Sorted sequence is 39, 53, 72, 81, 95.

3.3 Quicksort

Quicksort ဟာ divide and conquer approach ကိုအသုံးပြုထားပြီး performance အင်မတန်မြန်ပါတယ်။ အလုပ်လုပ်ပုဂ္ဂလဲ အင်မတန်ရှိရှင်းပါတယ်။ ပေးထားတဲ့ တန်ဖိုးအစဉ်ထဲက ကျပန်းကိန်းလုံးတစ်လုံးကိုရွေးယူလိုက်ပြီး အဲဒီကိန်းအပေါ်မူတည်လို့ သူ့ထက်ပိုင်ယောက်တဲ့ကိန်းတွေသီးခြား sorting စီ၊ သူ့ထက်ပိုင်ယောက်တဲ့ ကိန်းတွေသီးခြား sorting စီတာဖြစ်ပါတယ်။ Time complexity အနေနဲ့ $O(n \log n)$ ရှိပါတယ်။ Quicksort

algorithm ကို Figure 3.4 မှာ တွေ့သြပါ။ Algorithm အရ sequence ထဲက ကျပန်းကိန်းတစ်လုံးကို အရင်ယူလိုက်ပါတယ်။ ပြီးတဲ့နောက် အဲဒီ sequence ထဲက ကျပန်းတန်ဖိုးထက်ပိုင်ယ်တဲ့ ကိန်းတွေကို S_1 အဖြစ်စုံ၊ ညီတဲ့ ကိန်းတွေကို S_2 အဖြစ်စုံ၊ ပိုကြီးတဲ့ ကိန်းတွေကို S_3 အဖြစ်စုံ။ ပြီးတဲ့နောက် S_1 နဲ့ S_3 နှစ်ခုထဲ ကိန်းတစ်လုံးတည်းသော်လည်းကောင်း၊ တစ်လုံးမှုမကျန်တော့တဲ့ အထိသော်လည်းကောင်း ရှေ့ကလို ထပ်ခါတလဲလဲ လုပ်သွားရပါမယ်။ အကုန်ပြီးတဲ့ အခါ sorting စီထားတဲ့ sequence ကို ချရေးပါမယ်။

```

procedure QUICKSORT( $S$ ):
    if  $S$  contains at most one element then return  $S$ 
    else
        begin
            choose an element  $a$  randomly from  $S$ :
            let  $S_1$ ,  $S_2$ , and  $S_3$  be the sequences of elements in  $S$  less
            than, equal to, and greater than  $a$ , respectively;
            return (QUICKSORT( $S_1$ ) followed by  $S_2$  followed by
                QUICKSORT( $S_3$ ))
        end

```

Figure 3.4 Quicksort Algorithm

Example 3.4. Sort the sequence $S = \{6, 9, 3, 1, 2, 7, 1, 8, 3\}$ using quicksort.

Solution:

$\text{QUICKSORT}(S = \{6, 9, 3, 1, 2, 7, 1, 8, 3\})$

Choose $a = 9$, then return

$$S_1 = \{6, 3, 1, 2, 7, 1, 8, 3\}, S_2 = \{9\}, S_3 = \{ \ }$$

Choose $a = 6$, then return

$$S_1 = \{3, 1, 2, 1, 3\}, S_2 = \{6\}, S_3 = \{7, 8\}, \{9\}, \{ \ }$$

Choose $a = 2$, then return

$$S_1 = \{1, 1\}, S_2 = \{2\}, S_3 = \{3, 3\}, \{6\}, S_3 = \{7, 8\}, S_3 = \{7, 8\}, \{9\}, \{ \ }$$

Choose $a = 1$, then return

$$S_1 = \{ \}, S_2 = \{1, 1\}, S_3 = \{ \}, \{2\}, S_3 = \{3, 3\}, \{6\}, S_3 = \{7, 8\}, S_3 = \{7, 8\}, \{9\}, \{ \}$$

Choose $a = 3$, then return

$$\{ \}, \{1, 1\}, \{ \}, \{2\}, \{ \}, \{3, 3\}, \{ \}, \{6\}, S_1 = \{ \}, S_2 = \{7\}, S_3 = \{8\}, \{9\}, \{ \}$$

Choose $a = 7$, then return

$$\{ \}, \{1, 1\}, \{ \}, \{2\}, \{ \}, \{3, 3\}, \{ \}, \{6\}, S_1 = \{ \}, S_2 = \{7\}, S_3 = \{8\}, \{9\}, \{ \}$$

Since each set contains only one element, return

$$\{ \}, \{1, 1\}, \{ \}, \{2\}, \{ \}, \{3, 3\}, \{ \}, \{6\}, \{ \}, \{7\}, \{8\}, \{9\}, \{ \}$$

∴ The sorted sequence is 1, 1, 2, 3, 3, 6, 7, 8, 9

3.4 Order Statistics

Dataset တစ်ခုထက် k ကြိမ်မြောက်အငယ်ဆုံးကိန်းရှာတာကို order statistics လိုပေါ်ပါတယ်။ ဥပမာအနေနဲ့ 7, 4, 2, 4 ဆိုတဲ့ sequence ထဲမှာ 4 သည် ခုတိယနဲ့ စတုတွေမြောက် အငယ်ဆုံးကိန်း ဖြစ်ပါတယ်။ ဘယ်လိုကနေဘယ်လို ဖြစ်သွားတာလဲဆိုတော့ ကိန်းလေးခုလုံးကို ငယ်စီကြီးလိုက်စီလိုက်ပြီး 2, 4, 4, 7 ထဲက 4 သည် ဘယ်နှေရာမြောက်မှာ တွေ့ရသလဲဆိုတာကို ချရေးလိုက်တာသာ ဖြစ်ပါတယ်။ ဒါလို ကိန်းလေး ခုနဲ့ဆိုရင်တော့ ဘယ်နှေကြိမ်မြောက် အငယ်ဆုံးကိန်းကို ဘယ်လိုရှာရှာ လွယ်ပေမဲ့ ကိန်းလုံးဆယ်ချို့ ရာချို့ပေးထားမယ်ဆိုရင် သိပ်လွယ်တော့မှာ မဟုတ်ပါဘူး။

သေချာတာတစ်ခုက ရှိသမျှ ဆယ်ချို့ ရာချို့သော ကိန်းအားလုံးကို စာဖတ်သူအနေနဲ့ အချိန်ကုန်ခံပြီး sorting စီချင်မှာ မဟုတ်ပါဘူး။ ဒါကြောင့် ဒီလိုကိန်းလုံးအမြောက်အမြားထဲကနေ ဘယ်နှေကြိမ်မြောက် အငယ်ဆုံးဘယ်ကိန်းဆိုပြီး ရွေးဖို့ time complexity $O(n)$ ပဲလိုတဲ့ နည်းလမ်းကို Figure 3.5 က algorithm မှာကြည့်ပါ။

```

procedure SELECT( $k, S$ ):
    if  $|S| < 50$  then
        begin
            sort  $S$ ;
            return  $k$ th smallest element in  $S$ 
        end
    else
        begin
            divide  $S$  into  $\lfloor |S|/5 \rfloor$  sequences of 5 elements each
            with up to four leftover elements;
            sort each 5-element sequence;
            let  $M$  be the sequence of medians of the 5-element sets;
             $m \leftarrow \text{SELECT}(\lfloor |M|/2 \rfloor, M)$ ;
            let  $S_1$ ,  $S_2$ , and  $S_3$  be the sequences of elements in  $S$  less
            than, equal to, and greater than  $m$ , respectively;
            if  $|S_1| \geq k$  then return  $\text{SELECT}(k, S_1)$ 
            else
                if  $(|S_1| + |S_2| \geq k)$  then return  $m$ 
                else return  $\text{SELECT}(k - |S_1| - |S_2|, S_3)$ 
        end
    end

```

Figure 3.5 k -th smallest element ရှာတဲ့ algorithm

Dataset အနေနဲ့ ပေးထားတဲ့ ကိန်းလုံးသည် အလုံး (၅၀) ထက်နည်းမယ်ဆိုရင် ခုနက္ခာပမာလိုပဲ sorting စီပြီး လိုချင်တဲ့ကိန်းကို လွယ်လင့်တကူ ရွေးပေးလိုက်ရှုပါပဲ။ အလုံး (၅၀) နဲ့အထက်ဆိုရင်တော့ ပေးထားတဲ့ ကိန်းတွေကို ငါးလုံးတစ်စုစု ခွဲပေးရပါမယ်။ အဲဒီငါးလုံးအစုလေးတွေကို sorting စီပေးရပါမယ်။ ပြီးတဲ့ နောက် တစ်စုစုရဲ့ အလယ်တည့်တည့်က ကိန်းလုံး median လေးတွေကို M array ထဲမှာ စုပြုး sorting စီပေးရပါမယ်။ ဆက်ပြီးတော့ M array ထဲက အလယ်ကိန်းကို ထပ်ယူပြီး m လို့ သတ်မှတ်လိုက်ပါမယ်။ တစ်ဆက် တည်းမှာပဲ ကျေန်ကိန်းတွေကိုလဲ m ထက်ယောက်တဲ့အစု S_1 , ညီတဲ့အစု S_2 , ပိုကြီးတဲ့အစု S_3 တွေအဖြစ် စုရပါမယ်။ တကယ်လို S_1 ထဲမှာ ရွေးယူချင်တဲ့ကိန်းထက် တန်ဖိုးများများ ပိုရှိတယ်ဆိုရင် သာမန်အတိုင်း ရွေးယူလိုက်ရှုပါပဲ $\text{SELECT}(k, S_1)$ ။ တကယ်လို S_1, S_2 နှစ်ခုပေါင်းမှ ရွေးယူချင်တဲ့ကိန်းထက် တန်ဖိုးများတယ်ဆိုရင် m ကိုပဲယူ။ ကျေန်တဲ့အခြေအနေတွေမှာဆိုရင်တော့ $\text{SELECT}(k - |S_1| - |S_2|, S_3)$ အတိုင်းယူ။

Example 3.5. Find the 15th smallest element in the given list S which contains 2, 6, 4, 9, 12, 21, 9, 4, 32, 10, 17, 14, 13, 25, 31, 8, 30, 29, 24, 21, 11, 17, 7, 8, 2, 9, 1, 15, 30, 32, 27, 45, 16, 27, 20, 12, 16, 4, 2, 13, 4, 9, 5, 13, 32, 29, 23, 14, 9, 21.

Solution:

Using selection algorithm, $|S| = 50, k = 15$

$$\begin{aligned} S_1 &= \{2, 6, 4, 9, 12\}, & S_2 &= \{21, 9, 4, 32, 10\}, \\ S_3 &= \{17, 14, 13, 25, 31\}, & S_4 &= \{8, 30, 29, 24, 21\} \\ S_5 &= \{11, 17, 7, 8, 2\}, & S_6 &= \{9, 1, 15, 30, 32\} \\ S_7 &= \{27, 45, 16, 27, 20\}, & S_8 &= \{12, 16, 4, 2, 13\} \\ S_9 &= \{4, 9, 5, 13, 32\}, & S_{10} &= \{29, 23, 14, 9, 21\} \end{aligned}$$

After sorting each sequence,

$$\begin{aligned} S_1 &= \{2, 4, 6, 9, 12\}, & S_2 &= \{4, 9, 10, 21, 32\}, \\ S_3 &= \{13, 14, 17, 25, 31\}, & S_4 &= \{8, 21, 24, 29, 30\} \\ S_5 &= \{2, 7, 8, 11, 17\}, & S_6 &= \{1, 9, 15, 30, 32\} \\ S_7 &= \{16, 20, 27, 27, 45\}, & S_8 &= \{2, 4, 12, 13, 16\} \\ S_9 &= \{4, 5, 9, 13, 32\}, & S_{10} &= \{9, 14, 21, 23, 29\} \end{aligned}$$

The sequence of medians is

$$M = \{6, 10, 17, 24, 8, 15, 27, 12, 9, 21\}$$

After recursive call $SELECT(\lceil \frac{M}{2} \rceil, M)$,

$$M = \{6, 8, 9, 10, 12, 15, 17, 21, 24, 27\}$$

We get $m = 12$

$$S_1 = \{2, 6, 4, 9, 9, 4, 10, 8, 11, 7, 8, 2, 9, 1, 4, 2, 4, 9, 5, 9\}, \quad |S_1| = 20$$

$$S_2 = \{12, 12\}, \quad |S_2| = 2$$

$$S_3 = \begin{matrix} 21, 32, 17, 14, 13, 25, 31, 30, 29, 24, 21, \\ 17, 15, 30, 32, 27, 45, 16, 27, 20, 16, 13, 13, 32, 29, 23, 14, 21 \end{matrix}, \quad |S_3| = 28$$

Since $|S_1| \geq k$, then

Recursive call to $SELECT(15, 20)$

$$S = \{1, 2, 2, 2, 4, 4, 4, 4, 5, 6, 7, 8, 8, 9, 9, 9, 9, 9, 10, 11\}$$

Return 9 as 15th smallest element.

Analysis of Algorithms

Comprehensive Theory Notes & Workbook for UCS Students

In this book,

contains precise, yet straightforward conceptual notes, definitions and trace exercises to practice are presented for these topics in the simplest way -

- Models of Computation
- Divide & Conquer Approach
- Dynamic Programming
- Advanced Sorting
- Order Statistics

The materials delivered by this book primarily aims to be used by UCS students for their studies and for exam preparations, this can also be used as a foundation for further studies on data structures and algorithms.

