Amortized analysis

# Data Structures - Assignment 2

Winter 2022

## 1. Question

[10 marks] In the Locate algorithm shown in class for the resizable array solution that requires a space overhead of O(√n), we saw the following formula:

$$\sum_{j=0}^{k-1} 2^{\lfloor j/2 \rfloor} = \begin{cases} 2 \times (2^{k/2} - 1) & \text{if } k \text{ is even;} \\ 3 \times 2^{(k-1)/2} - 2 & \text{otherwise.} \end{cases}$$

- [5 marks] Prove by induction that this equality holds for any positive integer k.
- [5 marks] Suppose that you were not given this equality. Instead, derive this formula from scratch. Show your work.

**Solution:** Formula is derived first and then proved by induction:

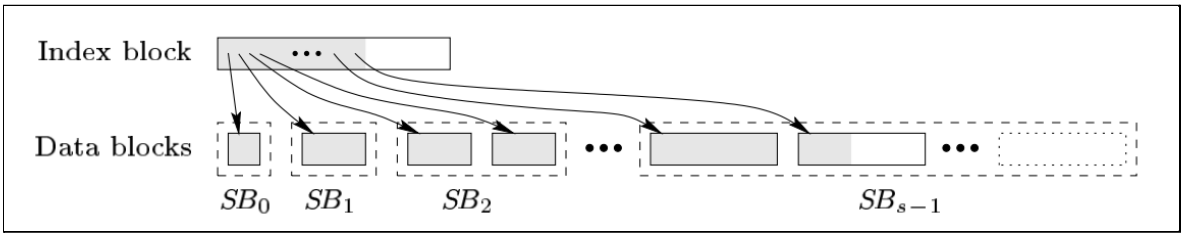


*Figure 1: A generic snapshot of the data structure [6]*

Data structures components:

- Index Block
- Data Blocks ($DB_0$, $DB_1$, $DB_2$…$DB_{d-1}$)
- Super Blocks ($SB_0$, $SB_1$, $SB_2$…$SB_{s-1}$)

Where:

- (n) is the number of elements.
- (d) is the number of non-empty data blocks, and the number of empty data blocks is always 1 or 0.
- (s) is the number of super-blocks.

Overview:

- Data blocks are grouped into superblocks (conceptually).
- Data Blocks in the same superblock are of the same size.
- When super block $SB_k$ is fully allocated, it consists of 2^floor(k/2) data blocks, each of size 2^ceil(k/2); size of SBk = 2^floor(k/2) * 2^ceil(k/2) = 2^k
- Only the last superblock, $SB_{k-1}$ might not be fully allocated.

Locate Algorithm:

- Let r denote the binary representation of (i + 1)
- The desired element i is the element e of the data block b and superblock k, where
    - k = |r| - 1,

- b is the floor(k/2) bits of r immediately after the leading 1-bit and
- e is the last ceil(k/2) bits of r.

Since knowing b and k alone does not suffice to find p, the number of data blocks in the super block before $B_k$, we have:

$$\Rightarrow p = \sum_{j=0}^{k-1} 2^{\lfloor j/2 \rfloor}$$

considering even and odd values seperately

Let $f(k) = \sum_{j=0}^{k-1} 2^{\lfloor j/2 \rfloor}$

Then $f(2k) = \sum_{j=0}^{2k-1} 2^{\lfloor j/2 \rfloor}$

$$= \sum_{j=0}^{k-1} \left(2^{\lfloor 2j/2 \rfloor} + 2^{\lfloor (2j+1)/2 \rfloor}\right)$$

$$= \sum_{j=0}^{k-1} \left(2^j + 2^j\right)$$

$$= 2\sum_{j=0}^{k-1} 2^j \quad \text{—— (1)}$$

$$= 2(2^k - 1) \quad \left(\text{wkt } \sum_{k=0}^{n} 2^k = 2^{n+1} - 1\right)$$

and $f(2k+1) = f(2k) + 2^k$ —— (2)

$$= 2(2^k - 1) + 2^k$$

$$= 3(2^k) - 2$$

Since $k = \frac{2k}{2} = \left\lfloor \frac{2k+1}{2} \right\rfloor$

we have,

$$\sum_{j=0}^{k-1} 2^{\lfloor j/2 \rfloor} = \begin{cases} 2 \times (2^{k/2} - 1) & \text{if } k \text{ is even;} \\ 3 \times 2^{(k-1)/2} - 2 & \text{otherwise.} \end{cases}$$

**Proof by induction** that this equality holds for any positive integer k.

For the sake of simplicity.

for splitting even & odd cases, consider $f(2n)$ & $f(2n+1)$

w.k.t $f(2n) = \sum_{j=0}^{2n-1} 2^{\lfloor j/2 \rfloor} = 2\sum_{j=0}^{n-1} 2^j$ — (from (1))

$f(2n) = 2\sum_{j=0}^{n-1} 2^j = 2 \times (2^{2n/2} - 1)$ — (3)

Similarly, $f(2n+1) = 2\sum_{j=0}^{n-1} 2^j + 2^n = 3 \times 2^{(2n+1-1)/2} - 2$ — (4)

→ from (2)

PROOF BY INDUCTION for (3) — Even Case

i) Base case: Verify of $n = 1$

$2(2^0) = 2(2^1 - 1)$

$2 = 2$ (True as expected)

2) Inductive step: suppose it is true of $n = k$

we have: $f(2k) = \sum_{j=0}^{k-1} 2^j = 2 \times (2^k - 1)$ — (5)

3) To prove that it is true for $n = k+1$

$f(2(k+1)) = 2\sum_{j=0}^{k} 2^j$ → To prove → $2 \times (2^{k+1} - 1)$

we know that $\sum_{k=0}^{n} 2^k = 2^{n+1} - 1$ (proof added seperately at the end)

Hence, after substitution

$f(2(k+1)) = 2 \times (2^{k+1} - 1)$, thus proved by induction

**Proof by induction** continued:

PROOF BY INDUCTION for (4) — Odd Case

1) Base Case: Verify for $n=1$

$2(2^0)+2^1 = 3\times 2^1 - 2$

$4 = 4$ (True as expected)

2) Inductive step: Suppose it is true for $n=k$

$f(2k+1) = 2\sum_{j=0}^{k-1} 2^j + 2^k = 3\times 2^k - 2$

3) To prove that it is true for $n=k+1$

$f(2(k+1)+1) = 2\sum_{j=0}^{k} 2^j + 2^{k+1}$ → To Prove → $3\times 2^{k+1} - 2$

we know that

$\sum_{k=0}^{n} 2^k = 2^{n+1} - 1$ (proof added seperately at the end)

Hence, after substitution

$f(2(k+1)+1) = 2\times(2^{k+1}-1) + 2^{k+1}$

$= 2(2^{k+1}) - 2 + 2^{k+1}$

$= 3(2^{k+1}) - 2$, Thus proved by induction

**Note**:

- The derivation of the formula from scratch is done first, followed by the proof by induction for reference in the proof and a better flow of answers.
- The proof for the geometric series referred to in the above induction proof is proved in the next page.

Proof for summation of geometric series used earlier:

Let $f(k) = \sum_{j=0}^{k-1} 2^{\lfloor j/2 \rfloor}$ (optional Extra work)

analysing the series for $k=6$

$f(6) = 2^0 + 2^0 + 2^1 + 2^1 + 2^2 + 2^2$

or $f(n) = 2(2^0 + 2^1 + 2^2 + \dots + 2^{n-1})$

{ also supported by the above proof
$f(2k) = 2\sum_{j=0}^{k-1} 2^j$ and $f(2k+1) = 2\sum_{j=0}^{k-1} 2^j + 2^k$ }

Irrespective, proof for summing a general geometric series:

$$1 + x + x^2 + \dots + x^{n-1} = \frac{x^n - 1}{x - 1}$$

In this case, $x = 2$

Hence, we have $2^0 + 2^1 + 2^2 + \dots + 2^{n-1} = 2^n - 1$

Proof by INDUCTION:

1) Base case: verify for $n=1$

$2^0 = 1 = 2^1 - 1$ (True as expected)

2) Inductive step: Suppose it is true for $n = k$

we have: $2^0 + 2^1 + 2^2 + \dots + 2^{(k-1)} = 2^k - 1$ —(6)

3) To prove that it is true for $n = k+1$

we have: $2^0 + 2^1 + 2^2 + \dots + 2^k$

$= 2^0 + 2^1 + 2^2 + \dots + 2^{k-1} + 2^k$

$= 2^k - 1 + 2^k$ (from (6))

$= 2 \cdot 2^k - 1$

$= 2^{k+1} - 1$, thus completing the inductive proof.

## 2. Question

[10 marks] Prove the Ω(n lg n) lower bound on sorting under the binary decision tree model. That is, prove that, given a sequence of n elements, the worst-case running time of any comparison-based algorithm that sorts these elements is Ω(n lg n).

Hint: Stirling's approximation may be helpful. It is on page 57 of the CLRS book. You can also find it in the following course note that I wrote for CSCI 3110 [3]

## Solution:

Binary Decision Tree Model (Getting Started):

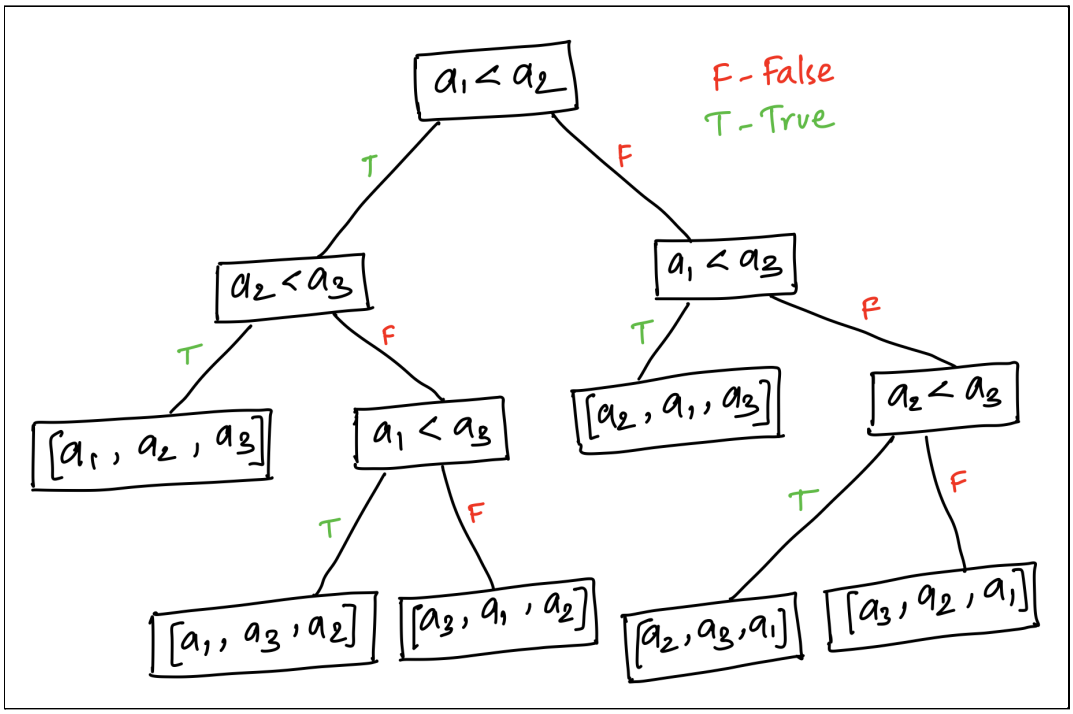


*Figure 2: Binary Decision Tree with all possible permutations for a list of 3 items.*

To start with, consider a binary tree of height k with 2^k leaves, as each leaf can translate to a different path from the root of the tree k to the very bottom. At each leaf, since it's a binary tree, there are two possible paths that it can lead to. With k number of choices, each of those can increase at most by a factor of 2 as seen in Figure 2.

Hence there are at most 2^k paths from the root to the bottom of the tree and at most 2^k leaves, and the height of the tree k is log(n), where n is the number of nodes/leaves in the given binary tree.
The worst-case number of comparisons corresponds to a max tree height; therefore, lower bound on height -> lower bound on sorting.

Adding to it, for a decision tree of height k, the number of paths (permutations) is n!
$n! \leq 2^k$

so, k ≥ lg(n!) ≥ Ω(n*lgn); Pre-work:

$$2^k \geq n!$$

$$k \geq \log(n!)$$

$$= \log(n(n-1)(n-2)(n-3)\dots(2))$$

$$= \log n + \log(n-1) + \log(n-2) + \dots + \log 2$$

$$= \sum_{i=2}^{n} \log i$$

$$= \sum_{i=2}^{n} \log i + \sum_{i=n/2}^{n} \log i$$

$$= 0 + \sum_{i=n/2}^{n} \log(n/2)$$

$$= (n/2)\log(n/2)$$

$$= \Omega(n \log n)$$

Using Stirling's Approximation:

So far, we know that the height of decision tree is $\Omega(\log n!)$ — (1)

From Stirling's approximation, w.k.t

$$n! = \sqrt{2\pi n}\left(\frac{n}{e}\right)^n \left(1 + \Theta\left(\frac{1}{n}\right)\right) \quad \text{— (2)}$$

Taking log on both sides

$$\log(n!) = \log\left(\sqrt{2\pi n}\left(\frac{n}{e}\right)^n (1 + \Theta(1/n))\right)$$

$$= \log(2\pi n)^{1/2} + \log(n/e)^n + \log(1 + \Theta(1/n))$$

$$= \frac{1}{2}\log(2\pi n) + n\log(n/e) + \log(1 + \Theta(1/n))$$

$$\geq n\log n - n\log e$$

$$= \Omega(n \log n) \quad \text{— (3)}$$

The above Equation (2) is from Stirling's approximation from CLRS Page 57, Chapter 3.2 [2].

## 3. Question

[15 marks] In class we learned that the space cost of van Emde Boas trees satisfies:

$$S(u) = (\sqrt{u} + 1)S(\sqrt{u}) + \sqrt{u}$$

- [8 marks] Use the substitution method to prove that S(u) = O(u).
  Hint: Review Section 4.3 of the CLRS book if you are not familiar with the substitution method. Pages 85-86 on “Subtleties” are particularly helpful [4].
- [7 marks] Prove that S(u) = Ω(u).

## Solution:

Before proving S(u) = O(u), the space for the universe structure u.
Analyzing the space taken by the data structures by looking at the space taken by different components of van Emde Boas data structure, we have:

- The space for storing MIN and MAX elements is O(1), assuming that each integer fits in a single word.
- Since there are u^(1/2) sub-widgets, the total space taken by sub-widgets is S(u^(1/2))*u^(1/2).
- And a summary widget of space S(u^(1/2)).
- Finally, an array of pointers, i.e., One pointer for each children (sub-widgets), takes up u^(1/2) space.

Adding all of these together, we have S(u) = (u^(1/2) + 1) S(u^(1/2)) + u^(1/2)

Using the substitution method to prove that S(u) = O(u) and to prove S(u) = Ω(u)
Guessing the form of a solution for the above recurrence relation to find an upper and a lower bound, we have:

Continued on the next page.

Let's say, $S(k) < c_1 k - c_2$; such that $k < u$ —(1)
Substituting upper bound into the recurrence
we have $S(u) < (\sqrt{u}+1)(c_1\sqrt{u} - c_2) + \sqrt{u}$

$= c_1 u - c_2\sqrt{u} + c_1\sqrt{u} - c_2 + \sqrt{u}$

$= c_1 u - c_2 - \sqrt{u}(c_2 - c_1 - 1)$ —(2)

By choosing $c_1$ and $c_2$ such that $(c_2 - c_1 - 1) > 0$
Equation (1) is satisfied, Hence the space $S(u)$ is $O(u)$

Similarly, assumming $S(k) > ck$; such that $k < u$ —(3)
substituting the lower bound into recurrence
we have $S(u) > (\sqrt{u}+1)(c\sqrt{u}) + \sqrt{u}$

$= cu + (c+1)\sqrt{u}$

$> cu$ —(4)

Equation (3) is satisfied, Hence the space $S(u)$ is $\Omega(u)$

---

## 4. Question

[15 marks] Suppose that we have the following variant of van Emde Boas trees: Let W be a widget whose universe size is u (i.e., W stores u bits). Then, instead of using √u sub widgets each of size √u to represent Was done in class, we use u^1/3 sub widgets, and the universe size of each subwidget is u^2/3. The size of the summary widget of W is then set to u^1/3 so that it still has exactly one bit for each subwidget.

The above is done for every widget in the van Emde Boas tree when building it recursively. For simplicity, assume that u^1/3 and u^2/3 are both integers.

- [6 marks] To perform Member, Successor, Insert and Delete operations over this variant, what changes need to make to the pseudocode shown in class?
- [9 marks] Analyze the running time of each operation over this variant.

## Solution:

Referring to the pseudo-code of the membership query discussed in class and Chapter 20 (page 550) in the book Introduction to Algorithms - CLRS [2] as seen in Figure 4.
The pseudo-code does not require any changes besides the difference in the division of widgets. Irrespective, MIN and MAX take O(1) or constant.

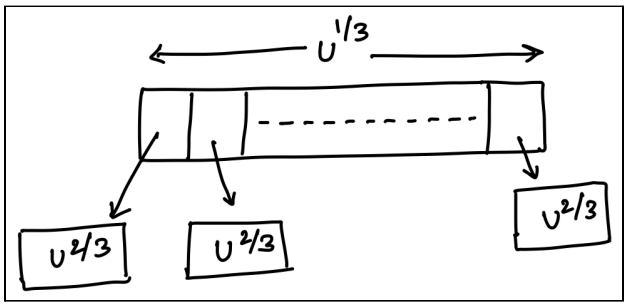


*Figure 3:* van Emde Boas trees*, U^1/3; U^2/3 variant*

```
VEB-TREE-MEMBER(V, x)
1  if x == V.min or x == V.max
2       return TRUE
3  elseif V.u == 2
4       return FALSE
5  else return VEB-TREE-MEMBER(V.cluster[high(x)], low(x))
```

*Figure 4: Recursive function Pseudo-code for Membership, CLRS [2]*

The recurrence and runtime analysis for MEMBERSHIP, SUCCESSOR, PREDECESSOR, INSERT and DELETE are as follows:

Continued on the next page.

MEMBERSHIP:

For the $u^{1/2}$ variant, we know that, the recurrence is

$$T(u) = T(u^{1/2}) + O(1)$$
$$\Rightarrow O(\lg \lg u)$$
$$\text{i.e, } O(\log_2 \log_2 u) \quad \text{—(1)}$$

For the $u^{1/3}$ submidgets
& $u^{2/3}$ universe size of each sub-midget

$$T(u) = T(u^{2/3}) + O(1)$$
$$\Rightarrow O(\log_{3/2} \log_2 u) \quad \text{—(2)}$$

the order of growth almost the same as (1)

Similarly, for the $u^{1/2}$ variant, the improved version of recurrence for INSERT, DELETE, SUCCESSOR & PREDECESSOR the run-time is $O(\lg \lg u)$ with recurrence $T(u^{1/2}) + O(1)$ with division of the widgets being the only difference, the recurrence is $T(u) = T(u^{2/3}) + O(1)$. —(3)

$\hookrightarrow T(u) = \max\{T(u^{1/3}), T(u^{2/3})\} + O(1)$

and the run-time is $O(\log_{3/2} \log_2 u)$ —(4)

or simply $O(\lg \lg u)$

The recurrence and runtime solution(s) for the 5 operations referred to the above analysis is from the class and CLRS page 550 to 556 Chapter 20.

## References


[1] "Amortized analysis," Wikipedia, Nov. 04, 2021. https://en.wikipedia.org/wiki/Amortized_analysis (accessed Jan. 15, 2022).

[2] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. 2009. Introduction to Algorithms, Third Edition (3rd. ed.). The MIT Press.

[3] “Lecture 3 Notes, CSCI 3110,” web.cs.dal.ca. https://web.cs.dal.ca/~mhe/csci3110/handouts/lecture3.htm (accessed Jan. 29, 2022).

[4] “Substitution Method,” site.ebrary.com. http://site.ebrary.com/lib/dal/docDetail.action?docID=10397652 (accessed Jan. 28, 2022).

[5] “4.3 The substitution method for solving recurrences,” itfs.egloos.com. http://itfs.egloos.com/m/9663923 (accessed Jan. 29, 2022).

[6] Brodnik, Andrej & Carlsson, Svante & Demaine, Erik & Munro, J. & Sedgewick, Robert. (2000). Resizable Arrays in Optimal Time and Space. Lecture Notes in Computer Science. 10.1007/3-540-48447-7_4.