Introduction to Algorithms

chapter 1

exercises

> 1.1-2 Other than speed, what other measures of efficiency might one use in a real-world setting?

1. **Memory Usage (Space Complexity):** Efficient algorithms should use an optimal amount of memory. Minimizing the space complexity can be crucial, especially in environments with limited memory resources.
2. **Power Consumption:** In scenarios where energy efficiency is crucial (e.g., mobile devices, IoT devices), algorithms that consume less power are desirable. Low-power algorithms can contribute to longer battery life and reduced environmental impact.
3. **Scalability:** An algorithm's performance should scale gracefully as the input size or workload increases. Scalability is essential in applications dealing with large datasets or high traffic.
4. **Parallelization and Concurrency, Robustness and Fault Tolerance, Adaptability** and so on...

chapter 2

exercises

> 2.1-2 Rewrite the INSERTION-SORT procedure to sort into nonincreasing instead of nondecreasing order.

```
INSERTION-SORT(A)
    for j = 2 to A.length
        key = A[j]
        i = j - 1
        while i > 0 and A[i] < key
            A[i + 1] = A[i]
            i = i - 1
        A[i + 1] = key
```

> 2.1-3 Consider the ***searching problem***:

> **Input**: A sequence of n numbers $A = \langle a_1, a_2, \ldots, a_n \rangle$ and a value v.
>
> **Output:** An index i such that $v = A[i]$ or the special value NIL if v does not appear in A.
>
> Write pseudocode for ***linear search***, which scans through the sequence, looking for v. Using a loop invariant, prove that your algorithm is correct. Make sure that your loop invariant fulfills the three necessary properties.

```
LINEAR-SEARCH(A, v)
    for i = 1 to A.length
       if A[i] == v
            return i
    return NIL
```

> Consider sorting n numbers stored in array A by first finding the smallest element of A and exchanging it with the element in $A[1]$. Then find the second smallest element of A, and exchange it with $A[2]$. Continue in this manner for the first $n-1$ elements of A. Write pseudocode for this algorithm, which is known as ***selection sort***. What loop invariant does this algorithm maintain? Why does it need to run for only the first $n-1$ elements, rather than for all n elements? Give the best-case and worst-case running times of selection sort in Θ-notation.

- Pseudocode:

  ```
  n = A.length
  for i = 1 to n - 1
      minIndex = i
      for j = i + 1 to n
          if A[j] < A[minIndex]
              minIndex = j
      swap(A[i], A[minIndex])
  ```

- Loop invariant:

  At the start of the loop in line 1, the subarray $A[1..i-1]$ consists of the smallest $i-1$ elements in array A with sorted order.

- Why does it need to run for only the first $n-1$ elements, rather than for all n elements?

  After $n-1$ iterations, the subarray $A[1..n-1]$ consists of the smallest $i-1$ elements in array A with sorted order. Therefore, $A[n]$ is already the largest element.

- Running time: $\Theta(n^2)$.

> 2.2-4 How can we modify almost any algorithm to have a good best-case running time?

You can modify any algorithm to have a best case time complexity by adding a special case. If the input matches this special case, return the pre-computed answer.

> Rewrite the MERGE procedure so that it does not use sentinels, instead stopping once either array L or R has had all its elements copied back to A and then copying the remainder of the other array back into A.

```
MERGE(A, p, q, r)
    n1 = q - p + 1
    n2 = r - q
    let L[1..n1] and R[1..n2] be new arrays
    for i = 1 to n1
        L[i] = A[p + i - 1]
    for j = 1 to n2
        R[j] = A[q + j]
    i = 1
    j = 1
    for k = p to r
        if i > n1
            A[k] = R[j]
            j = j + 1
        else if j > n2
            A[k] = L[i]
            i = i + 1
        else if L[i] ≤ R[j]
            A[k] = L[i]
            i = i + 1
```

```
        else
            A[k] = R[j]
            j = j + 1
```

> 2.3-5 Referring back to the searching problem (see Exercise 2.1-3), observe that if the sequence A is sorted, we can check the midpoint of the sequence against v and eliminate half of the sequence from further consideration. The ***binary search*** algorithm repeats this procedure, halving the size of the remaining portion of the sequence each time. Write pseudocode, either iterative or recursive, for binary search. Argue that the worst-case running time of binary search is $\Theta(\lg n)$.

- Iterative:

  ```
  ITERATIVE-BINARY-SEARCH(A, v, low, high)
      while low ≤ high
          mid = floor((low + high) / 2)
          if v == A[mid]
              return mid
          else if v > A[mid]
              low = mid + 1
          else high = mid - 1
      return NIL
  ```

- Recursive:

  ```
  RECURSIVE-BINARY-SEARCH(A, v, low, high)
      if low > high
          return NIL
      mid = floor((low + high) / 2)
      if v == A[mid]
          return mid
      else if v > A[mid]
          return RECURSIVE-BINARY-SEARCH(A, v, mid + 1, high)
      else return RECURSIVE-BINARY-SEARCH(A, v, low, mid - 1)
  ```

  Each time we do the comparison of v with the middle element, the search range continues with range halved.

  The recurrence

  $$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1, \\ T(n/2) + \Theta(1) & \text{if } n > 1. \end{cases} \tag{24}$$

  The solution of the recurrence is $T(n) = \Theta(\lg n)$.

chapter 3

problems

3-4

a. Disprove, $n = O(n^2)$, but $n^2 \neq O(n)$.

b. Disprove, $n^2 + n \neq \Theta(\min(n^2, n)) = \Theta(n)$.

c. Prove, because $f(n) \geq 1$ after a certain $n \geq n_0$.

$$\begin{aligned} \exists c, n_0 : \forall n \geq n_0, 0 \leq f(n) \leq cg(n) \\ \Rightarrow 0 \leq \lg f(n) \leq \lg(cg(n)) = \lg c + \lg g(n). \end{aligned} \tag{25}$$

We need to prove that

$\lg f(n) \leq d \lg g(n)$.

We can find d,

$d = \frac{\lg c + \lg g(n)}{\lg g(n)} = \frac{\lg c}{\lg g(n)} + 1 \leq \lg c + 1$,

where the last step is valid, because $\lg g(n) \geq 1$.

d. Disprove, because $2n = O(n)$, but $2^{2n} = 4^n \neq O(2^n)$.

e. Prove, $0 \leq f(n) \leq cf^2(n)$ is trivial when $f(n) \geq 1$, but if $f(n) < 1$ for all n, it's not correct. However, we don't care this case.

f. Prove, from the first, we know that $0 \leq f(n) \leq cg(n)$ and we need to prove that $0 \leq df(n) \leq g(n)$, which is straightforward with $d = 1/c$.

g. Disprove, let's pick $f(n) = 2^n$. We will need to prove that

$\exists c_1, c_2, n_0 : \forall n \geq n_0, 0 \leq c_1 \cdot 2^{n/2} \leq 2^n \leq c_2 \cdot 2^{n/2}$,

which is obviously untrue.

h. Prove, let $g(n) = o(f(n))$. Then

$\exists c, n_0 : \forall n \geq n_0, 0 \leq g(n) < cf(n)$.

We need to prove that

$\exists c_1, c_2, n_0 : \forall n \geq n_0, 0 \leq c_1 f(n) \leq f(n) + g(n) \leq c_2 f(n)$.

Thus, if we pick $c_1 = 1$ and $c_2 = c + 1$, it holds.

chapter 4

exercises

> 4.1-5 Use the following ideas to develop a nonrecursive, linear-time algorithm for the maximum-subarray problem. Start at the left end of the array, and progress toward the right, keeping track of the maximum subarray seen so far. Knowing a maximum subarray $A[1..j]$, extend the answer to find a maximum subarray ending at index $j + 1$ by using the following observation: a maximum subarray $A[i..j+1]$, is either a maximum subarray of $A[1..j]$ or a subarray $A[i..j+1]$, for some $1 \leq i \leq j + 1$. Determine a maximum subarray of the form $A[i..j+1]$ in constant time based on knowing a maximum subarray ending at index j.

```
ITERATIVE-FIND-MAXIMUM-SUBARRAY(A)
    n = A.length
    max-sum = -∞
```

```
    sum = -∞
    for j = 1 to n
        currentHigh = j
        if sum > 0
            sum = sum + A[j]
        else
            currentLow = j
            sum = A[j]
        if sum > max-sum
            max-sum = sum
            low = currentLow
            high = currentHigh
    return (low, high, max-sum)
```

> 4.3-2 Show that the solution of $T(n) = T(\lceil n/2 \rceil) + 1$ is $O(\lg n)$.

We guess $T(n) \le c \lg(n - a)$,

$$\begin{aligned} T(n) & \le c \lg(\lceil n/2 \rceil - a) + 1 \\ & \le c \lg((n + 1)/2 - a) + 1 \\ & = c \lg((n + 1 - 2a)/2) + 1 \\ & = c \lg(n + 1 - 2a) - c \lg 2 + 1 & (c \ge 1) \\ & \le c \lg(n + 1 - 2a) & (a \ge 1) \\ & \le c \lg(n - a), \end{aligned} \tag{26}$$

> 4.3-3 We saw that the solution of $T(n) = 2T(\lfloor n/2 \rfloor) + n$ is $O(n \lg n)$. Show that the solution of this recurrence is also $\Omega(n \lg n)$. Conclude that the solution is $\Theta(n \lg n)$.

First, we guess $T(n) \le cn \lg n$,

$$\begin{aligned} T(n) & \le 2c \lfloor n/2 \rfloor \lg \lfloor n/2 \rfloor + n \\ & \le cn \lg(n/2) + n \\ & = cn \lg n - cn \lg 2 + n \\ & = cn \lg n + (1 - c)n \\ & \le cn \lg n, \end{aligned} \tag{27}$$

where the last step holds for $c \ge 1$.

Next, we guess $T(n) \ge c(n + a) \lg(n + a)$,

$$
\begin{aligned}
T(n) &\ge 2c(\lfloor n/2 \rfloor + a)(\lg(\lfloor n/2 \rfloor + a) + n \\
&\ge 2c((n-1)/2 + a)(\lg((n-1)/2 + a)) + n \\
&= 2c\frac{n-1+2a}{2}\lg\frac{n-1+2a}{2} + n \\
&= c(n-1+2a)\lg(n-1+2a) - c(n-1+2a)\lg 2 + n \\
&= c(n-1+2a)\lg(n-1+2a) + (1-c)n - (2a-1)c & (0 \le c < 1, n \ge \frac{(2a-1)c}{1-c}) \\
&\ge c(n-1+2a)\lg(n-1+2a) & (a \ge 1) \\
&\ge c(n+a)\lg(n+a),
\end{aligned} \tag{28}
$$

> Use a recursion tree to determine a good asymptotic upper bound on the recurrence $T(n) = 3T(\lfloor n/2 \rfloor) + n$. Use the substitution method to verify your answer.

- The subproblem size for a node at depth i is $n/2^i$.

  Thus, the tree has $\lg n + 1$ levels and $3^{\lg n} = n^{\lg 3}$ leaves.

  The total cost over all nodes at depth i, for $i = 0, 1, 2, \ldots, \lg n - 1$, is $3^i(n/2^i) = (3/2)^i n$.

$$
\begin{aligned}
T(n) &= n + \frac{3}{2}n + \left(\frac{3}{2}\right)^2 n + \cdots + \left(\frac{3}{2}\right)^{\lg n - 1} n + \Theta(n^{\lg 3}) \\
&= \sum_{i=0}^{\lg n - 1}\left(\frac{3}{2}\right)^i n + \Theta(n^{\lg 3}) \\
&= \frac{(3/2)^{\lg n} - 1}{(3/2) - 1} n + \Theta(n^{\lg 3}) \\
&= 2[(3/2)^{\lg n} - 1]n + \Theta(n^{\lg 3}) \\
&= 2[n^{\lg(3/2)} - 1]n + \Theta(n^{\lg 3}) \\
&= 2[n^{\lg 3 - \lg 2} - 1]n + \Theta(n^{\lg 3}) \\
&= 2[n^{\lg 3 - 1 + 1} - n] + \Theta(n^{\lg 3}) \\
&= O(n^{\lg 3}).
\end{aligned} \tag{29}
$$

- We guess $T(n) \le cn^{\lg 3} - dn$,

$$
\begin{aligned}
T(n) &= 3T(\lfloor n/2 \rfloor) + n \\
&\le 3 \cdot (c(n/2)^{\lg 3} - d(n/2)) + n \\
&= (3/2^{\lg 3})cn^{\lg 3} - (3d/2)n + n \\
&= cn^{\lg 3} + (1 - 3d/2)n,
\end{aligned} \tag{30}
$$

where the last step holds for $d \ge 2$.

> 4.4-3 Use a recursion tree to determine a good asymptotic upper bound on the recurrence $T(n) = 4T(n/2 + 2) + n$. Use the substitution method to verify your answer.

- The subproblem size for a node at depth i is $n/2^i$.

  Thus, the tree has $\lg n + 1$ levels and $4^{\lg n} = n^2$ leaves.

  The total cost over all nodes at depth i, for $i = 0, 1, 2, \ldots, \lg n - 1$, is $4^i(n/2^i + 2) = 2^i n + 2 \cdot 4^i$.

$$
\begin{aligned}
T(n) &= \sum_{i=0}^{\lg n - 1} (2^i n + 2 \cdot 4^i) + \Theta(n^2) \\
&= \sum_{i=0}^{\lg n - 1} 2^i n + \sum_{i=0}^{\lg n - 1} 2 \cdot 4^i + \Theta(n^2) \\
&= \frac{2^{\lg n} - 1}{2 - 1} n + 2 \cdot \frac{4^{\lg n} - 1}{4 - 1} + \Theta(n^2) \\
&= (2^{\lg n} - 1)n + \frac{2}{3}(4^{\lg n} - 1) + \Theta(n^2) \\
&= (n - 1)n + \frac{2}{3}(n^2 - 1) + \Theta(n^2) \\
&= \Theta(n^2).
\end{aligned} \tag{31}
$$

- We guess $T(n) \le c(n^2 - dn)$,

$$
\begin{aligned}
T(n) &= 4T(n/2 + 2) + n \\
&\le 4c[(n/2 + 2)^2 - d(n/2 + 2)] + n \\
&= 4c(n^2/4 + 2n + 4 - dn/2 - 2d) + n \\
&= cn^2 + 8cn + 16c - 2cdn - 8cd + n \\
&= cn^2 - cdn + 8cn + 16c - cdn - 8cd + n \\
&= c(n^2 - dn) - (cd - 8c - 1)n - (d - 2) \cdot 8c \\
&\le c(n^2 - dn),
\end{aligned} \tag{32}
$$

where the last step holds for $cd - 8c - 1 \geq 0$.

> 4.4-8 Use a recursion tree to give an asymptotically tight solution to the recurrence $T(n) = T(n-a) + T(a) + cn$, where $a \geq 1$ and $c > 0$ are constants.

- The tree has $n/a + 1$ levels.

  The total cost over all nodes at depth i, for $i = 0, 1, 2, \ldots, n/a - 1$, is $c(n - ia)$.

$$
\begin{aligned}
T(n) &= \sum_{i=0}^{n/a} c(n - ia) + (n/a)ca \\
&= \sum_{i=0}^{n/a} cn - \sum_{i=0}^{n/a} cia + (n/a)ca \\
&= cn^2/a - \Theta(n) + \Theta(n) \\
&= \Theta(n^2).
\end{aligned} \tag{33}
$$

- For $O(n^2)$, we guess $T(n) \leq cn^2$,

$$
\begin{aligned}
T(n) &\leq c(n-a)^2 + ca + cn \\
&\leq cn^2 - 2can + ca + cn \\
&\leq cn^2 - c(2an - a - n) \qquad (a > 1/2, n > 2a) \\
&\leq cn^2 - cn \\
&\leq cn^2 \\
&= \Theta(n^2).
\end{aligned} \tag{34}
$$

- For $\Omega(n^2)$, we guess $T(n) \geq cn^2$,

$$
\begin{aligned}
T(n) &\geq c(n-a)^2 + ca + cn \\
&\geq cn^2 - 2acn + ca + cn \\
&\geq cn^2 - c(2an - a - n) \qquad (a < 1/2, n > 2a) \\
&\geq cn^2 + cn \\
&\geq cn^2 \\
&= \Theta(n^2).
\end{aligned} \tag{35}
$$

> 4.4-9 Use a recursion tree to give an asymptotically tight solution to the recurrence $T(n) = T(\alpha n) + T((1 - \alpha)n) + cn$, where α is a constant in the range $0 < \alpha < 1$, and $c > 0$ is also a constant.

We can assume that $0 < \alpha \leq 1/2$, since otherwise we can let $\beta = 1 - \alpha$ and solve it for β.

Thus, the depth of the tree is $\log_{1/\alpha} n$ and each level costs cn. And let's guess that the leaves are $\Theta(n)$,

$$\begin{aligned} T(n) &= \sum_{i=0}^{\log_{1/\alpha} n} cn + \Theta(n) \\ &= cn \log_{1/\alpha} n + \Theta(n) \\ &= \Theta(n \lg n). \end{aligned} \tag{36}$$

We can also show $T(n) = \Theta(n \lg n)$ by substitution.

To prove the upper bound, we guess that $T(n) \le dn \lg n$ for a constant $d > 0$,

$$\begin{aligned} T(n) &= T(\alpha n) + T((1-\alpha)n) + cn \\ &\le d\alpha n \lg(\alpha n) + d(1-\alpha)n \lg((1-\alpha)n) + cn \\ &= d\alpha n \lg \alpha + d\alpha n \lg n + d(1-\alpha)n \lg(1-\alpha) + d(1-\alpha)n \lg n + cn \\ &= dn \lg n + dn(\alpha \lg \alpha + (1-\alpha)\lg(1-\alpha)) + cn \\ &\le dn \lg n, \end{aligned} \tag{37}$$

where the last step holds when $d \ge \frac{-c}{\alpha \lg \alpha + (1-\alpha)\lg(1-\alpha)}$.

We can achieve this result by solving the inequality

$$\begin{aligned} dn \lg n + dn(\alpha \lg \alpha + (1-\alpha)\lg(1-\alpha)) + cn &\le dn \lg n \\ \implies dn(\alpha \lg \alpha + (1-\alpha)\lg(1-\alpha)) + cn &\le 0 \\ \implies d(\alpha \lg \alpha + (1-\alpha)\lg(1-\alpha)) &\le -c \\ \implies d &\ge \frac{-c}{\alpha \lg \alpha + (1-\alpha)\lg(1-\alpha)}, \end{aligned} \tag{38}$$

To prove the lower bound, we guess that $T(n) \ge dn \lg n$ for a constant $d > 0$,

$$\begin{aligned} T(n) &= T(\alpha n) + T((1-\alpha)n) + cn \\ &\ge d\alpha n \lg(\alpha n) + d(1-\alpha)n \lg((1-\alpha)n) + cn \\ &= d\alpha n \lg \alpha + d\alpha n \lg n + d(1-\alpha)n \lg(1-\alpha) + d(1-\alpha)n \lg n + cn \\ &= dn \lg n + dn(\alpha \lg \alpha + (1-\alpha)\lg(1-\alpha)) + cn \\ &\ge dn \lg n, \end{aligned} \tag{39}$$

where the last step holds when $0 < d \le \frac{-c}{\alpha \lg \alpha + (1-\alpha)\lg(1-\alpha)}$.

We can achieve this result by solving the inequality

$$
\begin{aligned}
& dn\lg n + dn(\alpha\lg\alpha + (1-\alpha)\lg(1-\alpha)) + cn \ge dn\lg n \\
& \implies dn(\alpha\lg\alpha + (1-\alpha)\lg(1-\alpha)) + cn \ge 0 \\
& \implies d(\alpha\lg\alpha + (1-\alpha)\lg(1-\alpha)) \ge -c \\
& \implies 0 < d \le \frac{-c}{\alpha\lg\alpha + (1-\alpha)\lg(1-\alpha)},
\end{aligned} \tag{40}
$$

Therefore, $T(n) = \Theta(n\lg n)$.

problems

> 4-3 Give asymptotic upper and lower bounds for $T(n)$ in each of the following recurrences. Assume that $T(n)$ is constant for sufficiently small n. Make your bounds as tight as possible, and justify your answers.
>
> **a.** $T(n) = 4T(n/3) + n\lg n$.
>
> **b.** $T(n) = 3T(n/3) + n/\lg n$.
>
> **c.** $T(n) = 4T(n/2) + n^2\sqrt{n}$.
>
> **d.** $T(n) = 3T(n/3 - 2) + n/2$.
>
> **e.** $T(n) = 2T(n/2) + n/\lg n$.
>
> **f.** $T(n) = T(n/2) + T(n/4) + T(n/8) + n$.
>
> **g.** $T(n) = T(n-1) + 1/n$.
>
> **h.** $T(n) = T(n-1) + \lg n$.
>
> **i.** $T(n) = T(n-2) + 1/\lg n$.
>
> **j.** $T(n) = \sqrt{n}T(\sqrt{n}) + n$

a. By master theorem, $T(n) = \Theta(n^{\log_3 4})$.

b.

By the recursion-tree method, we can guess that $T(n) = \Theta(n\log_3\log_3 n)$.

We start by proving the upper bound.

Suppose $k < n \implies T(k) \le ck\log_3\log_3 k - k$, where we subtract a lower order term to strengthen our induction hypothesis.

It follows that

$$
\begin{aligned}
T(n) &\le 3(c\frac{n}{3}\log_3\log_3\frac{n}{3} - \frac{n}{3}) + \frac{n}{\lg n} \\
&\le cn\log_3\log_3 n - n + \frac{n}{\lg n} \\
&\le cn\log_3\log_3 n,
\end{aligned} \tag{41}
$$

if n is sufficiently large.

The lower bound can proved analogously.

c. By master theorem, $T(n) = \Theta(n^{2.5})$.

d. It is $\Theta(n \lg n)$. The subtraction occurring inside the argument to T won't change the asymptotics of the solution, that is, for large n the division is so much more of a change than the subtraction that it is the only part that matters. once we drop that subtraction, the solution comes by the master theorem.

e. By the same reasoning as part (b), the function is $O(n \lg n)$ and $\Omega(n^{1-\epsilon})$ for every ϵ and so is $\tilde{O}(n)$, see Problem 3-5.

f. We guess $T(n) \le cn$,

$$\begin{aligned} T(n) &= T(n/2) + T(n/4) + T(n/8) + n \\ &\le \frac{7}{8}cn + n \le cn. \end{aligned} \tag{42}$$

where the last step holds for $c \ge 8$.

g. Recall that χ_A denotes the indicator function of A. We see that the sum is

$T(0) + \sum_{j=1}^{n} \frac{1}{j} = T(0) + \int_1^{n+1} \sum_{j=1}^{n+1} \frac{\chi_{j,j+1}(x)}{j} dx.$

Since $\frac{1}{x}$ is monatonically decreasing, we have that for every $i \in \mathbb{Z}^+$,

$\text{sup\_}x \in (i, i+1) \sum\_j = 1^{n+1} \frac{\chi_{j,j+1}(x)}{j} - \frac{1}{x} = \frac{1}{i} - \frac{1}{i+1} = \frac{1}{i(i+1)}.$

Our expression for $T(n)$ becomes

$T(N) = T(0) + \int_1^{n+1} \left(\frac{1}{x} + O(\frac{1}{\lfloor x \rfloor(\lfloor x \rfloor + 1)})\right) dx.$

We deal with the error term by first chopping out the constant amount between 1 and 2 and then bound the error term by $O(\frac{1}{x(x-1)})$ which has an anti-derivative (by method of partial fractions) that is $O(\frac{1}{n})$,

$$\begin{aligned} T(N) &= \int_1^{n+1} \frac{dx}{x} + O(\frac{1}{n}) \\ &= \lg n + T(0) + \frac{1}{2} + O(\frac{1}{n}). \end{aligned} \tag{43}$$

This gets us our final answer of $T(n) = \Theta(\lg n)$.

h. We see that we explicity have

$$\begin{aligned} T(n) &= T(0) + \sum_{j=1}^{n} \lg j \\ &= T(0) + \int_1^{n+1} \sum_{j=1}^{n+1} \chi_{(j,j+1)}(x) \lg j dx. \end{aligned} \tag{44}$$

Similarly to above, we will relate this sum to the integral of $\lg x$.

$\text{sup\_}x \in (i, i+1) \sum\_j = 1^{n+1} \chi\_(j, j+1)(x) \lg j - \lg x = \lg(j+1) - \lg j = \lg\left(\frac{j+1}{j}\right).$

Therefore,

$$T(n) \leq \int_i^n \lg(x+2) + \lg x - \lg(x+1) dx$$

$$(1 + O(\frac{1}{\lg n}))\Theta(n \lg n). \tag{45}$$

i. See the approach used in the previous two parts, we will get $T(n) = \Theta(\frac{n}{\lg n})$.

j. Let i be the smallest i so that $n^{\frac{1}{2^i}} < 2$. We recall from a previous problem (3-6.e) that this is $\lg \lg n$ Expanding the recurrence, we have that it is

$$T(n) = n^{1-\frac{1}{2^i}} T(2) + n + n \sum_{j=1}^{i}$$

$$= \Theta(n \lg \lg n). \tag{46}$$