

Assignment 1 Solutions

1. Let $f : \mathbb{N} \rightarrow \mathbb{R}^+$ and $g : \mathbb{N} \rightarrow \mathbb{R}^+$ be two functions. For each of the following statements determine whether it is true or false. If it is true, give a proof, otherwise give a counterexample.

- (a) If $f(n) \in \Theta(g(n))$, then $2^{f(n)} \in \Theta(2^{g(n)})$,

Solution: False. Consider $f(n) = 2n$ and $g(n) = n$. Then $f(n) = 2n \in \Theta(n) = \Theta(g(n))$. But $2^{f(n)} = 2^{2n} = (2^n)^2 \notin \Theta(2^n) = \Theta(2^{g(n)})$.

- (b) if $f(n) \in \Theta(g(n))$ and $f(n), g(n) \geq 2$ for all n , then $\log_2 f(n) \in \Theta(\log_2 g(n))$,

Solution: True. By definition there are $c_1, c_2 > 0$ and an integer n_0 such that for all $n \geq n_0$, $c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n)$. Taking log from the inequality we obtain $\log_2 c_1 + \log_2 g(n) \leq \log_2 f(n) \leq \log_2 c_2 + \log_2 g(n)$. $\log_2 f(n), \log_2 g(n) \geq 1$ and the difference between $\log_2 f(n)$ and $\log_2 g(n)$ is bounded by a constant and we have $\log_2 f(n) \in \Theta(\log_2 g(n))$.

Formally, if $\log_2 c_1 \geq 0$ choose $d_1 = 1$, and otherwise choose $d_1 = \frac{1}{1 - \log_2 c_1} > 0$. We argue that for $n \geq n_0$, $\log_2 f(n) \geq d_1 \log_2 g(n)$. If $\log_2 c_1 \geq 0$, then $\log_2 f(n) \geq \log_2 c_1 + \log_2 g(n) \geq \log_2 g(n) = d_1 \log_2 g(n)$. Otherwise, we have $\log_2 f(n) \geq \log_2 c_1 + \log_2 g(n)$, hence $(1 - \log_2 c_1) \log_2 f(n) \geq \log_2 f(n) - \log_2 c_1 \geq \log_2 g(n)$ (remember that $\log_2 f(n) \geq 1$). Dividing both sides by $1 - \log_2 c_1 > 0$ yields $\log_2 f(n) \geq d_1 \log_2 g(n)$.

Similarly, if $\log_2 c_2 \leq 0$ choose $d_2 = 1$, and otherwise choose $d_2 = 1 + \log_2 c_2 > 0$. We argue that for $n \geq n_0$, $\log_2 f(n) \leq d_2 \log_2 g(n)$. If $\log_2 c_2 \leq 0$, then $\log_2 f(n) \leq \log_2 c_2 + \log_2 g(n) \leq \log_2 g(n) = d_2 \log_2 g(n)$. Otherwise, we have $\log_2 f(n) \leq \log_2 c_2 + \log_2 g(n) \leq (\log_2 c_2 + 1) \log_2 g(n) = d_2 \log_2 g(n)$ (remember that $\log_2 g(n) \geq 1$).

We have shown that for all $n \geq n_0$, $d_1 \log_2 g(n) \leq \log_2 f(n) \leq d_2 \log_2 g(n)$, hence $\log_2 f(n) \in \Theta(\log_2 g(n))$.

- (c) if $f(n) \in \Theta(g(n))$, then $\sum_{i=1}^n f(i) \in \Theta(\sum_{i=1}^n g(i))$,

Solution: True. By definition there are $c_1, c_2 > 0$ and an integer n_0 such that for all $n \geq n_0$, $c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n)$. If we had $n_0 = 1$, we could just add up the inequalities

$$\begin{aligned} c_1 \cdot g(1) &\leq f(1) \leq c_2 \cdot g(1) \\ c_1 \cdot g(2) &\leq f(2) \leq c_2 \cdot g(2) \\ &\vdots \\ c_1 \cdot g(n) &\leq f(n) \leq c_2 \cdot g(n) \end{aligned}$$

to obtain $c_1 \cdot \sum_{i=1}^n g(i) \leq \sum_{i=1}^n f(i) \leq c_2 \cdot \sum_{i=1}^n g(i)$, and hence $\sum_{i=1}^n f(i) \in \Theta(\sum_{i=1}^n g(i))$.

We, however, do not necessarily have the condition $n_0 = 1$. In general, if $n_0 \neq 1$. We know that $\sum_{i=1}^{n_0-1} f(i) > 0$ and that $\sum_{i=1}^{n_0-1} g(i)$ is some fixed number. Hence there are positive constants $d_1, d_2 > 0$ such that

$$d_1 \cdot \sum_{i=1}^{n_0-1} g(i) \leq \sum_{i=1}^{n_0-1} f(i) \leq d_2 \cdot \sum_{i=1}^{n_0-1} g(i),$$

let $e_1 = \min(c_1, d_1) > 0$ and $e_2 = \max(c_2, d_2) > 0$. Then for any $n \geq n_0$

$$\begin{aligned} \sum_{i=1}^n f(i) &= \sum_{i=1}^{n_0-1} f(i) + \sum_{i=n_0}^n f(i) \geq d_1 \cdot \sum_{i=1}^{n_0-1} g(i) + c_1 \sum_{i=n_0}^n g(i) \geq \\ &e_1 \cdot \sum_{i=1}^{n_0-1} g(i) + e_1 \sum_{i=n_0}^n g(i) = e_1 \sum_{i=1}^n g(i). \end{aligned}$$

Similarly,

$$\begin{aligned} \sum_{i=1}^n f(i) &= \sum_{i=1}^{n_0-1} f(i) + \sum_{i=n_0}^n f(i) \leq d_2 \cdot \sum_{i=1}^{n_0-1} g(i) + c_2 \sum_{i=n_0}^n g(i) \leq \\ &e_2 \cdot \sum_{i=1}^{n_0-1} g(i) + e_2 \sum_{i=n_0}^n g(i) = e_2 \sum_{i=1}^n g(i). \end{aligned}$$

Hence for all $n \geq n_0$ we have $e_1 \cdot \sum_{i=1}^n g(i) \leq \sum_{i=1}^n f(i) \leq e_2 \cdot \sum_{i=1}^n g(i)$, and thus $\sum_{i=1}^n f(i) \in \Theta(\sum_{i=1}^n g(i))$.

(d) if $(f(n) + g(n))^2 \in \Theta((2g(n))^2)$, then $f(n) \in \Theta(g(n))$.

Solution: False. Take $f(n) = 1$ and $g(n) = n$. Then $(f(n) + g(n))^2 = (1 + n)^2 = 1 + 2n + n^2 = \Theta(n^2) = \Theta(4n^2) = \Theta((2g(n))^2)$, but $f(n) = 1 \neq \Theta(n) = \Theta(g(n))$.

2. Give a Θ -approximation of the following sums:

(a) $\sum_{i=1}^n i \cdot 2^i$

Solution: $f(n) = n \cdot 2^n$. The ratio between $f(n+1)$ and $f(n)$ is

$$\frac{f(n+1)}{f(n)} = \frac{(n+1) \cdot 2^{n+1}}{n \cdot 2^n} > \frac{2^{n+1}}{2^n} = 2 > 1,$$

hence the sum is geometric-like, and $\sum_{i=1}^n i \cdot 2^i = \Theta(f(n)) = \Theta(n \cdot 2^n)$.

(b) $\sum_{k=5}^{4n} \frac{1}{3k+7}$

Solution: This sum looks like a variation of the harmonic sum, and indeed

$$\sum_{k=5}^{4n} \frac{1}{3k+7} = \frac{1}{22} + \frac{1}{25} + \frac{1}{28} + \dots + \frac{1}{12n+7} > \sum_{k=1}^{12n+7} \frac{1}{k} = \Theta(\log(12n+7)) = \Theta(\log n).$$

On the other hand

$$\begin{aligned} \sum_{k=5}^{4n} \frac{1}{3k+7} &= \frac{1}{3} \sum_{k=5}^{4n} \frac{1}{k+7/3} > \frac{1}{3} \sum_{k=5}^{4n} \frac{1}{k+3} = \frac{1}{3} \sum_{k=8}^{4n+3} \frac{1}{l} = \\ &= \frac{1}{3} \left(\Theta(\log(4n+3)) - \sum_{l=1}^7 \frac{1}{l} \right) = \Theta(\log n). \end{aligned}$$

Hence $\sum_{k=5}^{4n} \frac{1}{3k+7} = \Theta(\log n)$. Note that it is also possible to obtain the same result using integration.

(c) $\sum_{i=2}^n \sum_{j=1}^n \frac{\log i}{j}$

Solution: We have

$$\sum_{i=2}^n \sum_{j=1}^n \frac{\log i}{j} = \sum_{i=2}^n (\log i) \cdot \sum_{j=1}^n \frac{1}{j} = \left(\sum_{i=2}^n \log i \right) \cdot \left(\sum_{j=1}^n \frac{1}{j} \right).$$

$\sum_{i=2}^n \log i$ is arithmetic-like, and $\sum_{j=1}^n \frac{1}{j}$ is harmonic. Hence

$$\sum_{i=2}^n \sum_{j=1}^n \frac{\log i}{j} = \left(\sum_{i=2}^n \log i \right) \cdot \left(\sum_{j=1}^n \frac{1}{j} \right) = \Theta(n \log n) \cdot \Theta(\log n) = \Theta(n \log^2 n).$$

(d) $\sum_{k=1}^n k^k$

Solution: $f(n) = n^n$. The ratio between $f(n+1)$ and $f(n)$ is

$$\frac{f(n+1)}{f(n)} = \frac{(n+1)^{n+1}}{n^n} = \frac{(n+1)^n(n+1)}{n^n} > \frac{n^n(n+1)}{n^n} = n+1 \geq 2 > 1.$$

Hence the sum is geometric-like, and $\sum_{k=1}^n k^k = \Theta(f(n)) = \Theta(n^n)$.

3. Suppose that someone has developed an algorithm to solve a certain problem, which runs in time $T(n, m) \in \Theta(f(n, m))$, where n is the size of the input, and m is a parameter we are free to choose (we can choose it depending on n). In each case determine the value of the parameter $m(n)$ to achieve the (asymptotically) best running time. Justify your answer.

(a) $f(n, m) = \frac{n^3}{m} + m \cdot n$

Solution: Observe that if m is very small then the first term is very big, and we can make the whole expression smaller by increasing m . Similarly, if m is big, the second term dominates the expression, and we can decrease it by decreasing m . So for the “optimal” solution the two terms should be roughly the same. In this case $\frac{n^3}{m} = m \cdot n$ implies $m = n$, and yields $f(n, m) = \frac{n^3}{n} + n \cdot n = 2n^2 = \Theta(n^2)$.

We can check that n^2 is the (asymptotically) best result one can expect. For an arbitrary m , if $m < n$ then $f(n, m) > \frac{n^3}{m} > n^2$, and if $m \geq n$, then $f(n, m) > m \cdot n \geq n^2$.

(b) $f(n, m) = \log^3 m + \frac{2^n}{m}$

Solution: As before, we expect to achieve the best result when the terms are roughly equal. We are trying to solve the equation $\log^3 m = \frac{2^n}{m}$, or $m \log^3 m = 2^n$. The first approximation of the solution is $m = 2^n$. For this m we see that the left hand side is bigger than the right hand side by a factor of n^3 , so our second approximation is $m = \frac{2^n}{n^3}$, then

$$m \log^3 m = \frac{2^n}{n^3} \log^3 \frac{2^n}{n^3} = \frac{2^n}{n^3} (\log 2^n - \log n^3)^3 = \frac{2^n}{n^3} (n - 3 \log n)^3 = \Theta\left(\frac{2^n}{n^3}\right) = \Theta(2^n).$$

Hence this is an asymptotic solution to the equation. $m = \frac{2^n}{n^3}$ yields $f(n, m) = \log^3(2^n/n^3) + \frac{2^n}{2^n/n^3} = (\log 2^n - 3 \log n)^3 + n^3 = (n - 3 \log n)^3 + n^3 = \Theta(n^3)$. One can see that the same result is achieved with $m = 2^n$.

We now check that this is the (asymptotically) best result one can expect. For an arbitrary m , if $m < \frac{2^n}{n^3}$, then $f(m, n) > \frac{2^n}{m} > n^3$, and if $m \geq \frac{2^n}{n^3}$ then $f(n, m) > \log^3 m = (\log 2^n - 3 \log n)^3 = (n - 3 \log n)^3 = \Theta(n^3)$.

(c) $f(n, m) = \frac{8^n n^2}{m} + m \cdot 2^n + m^2$

Solution: As before we see that when m grows the first term decreases while the other two increase. So if initially m is very small, we should increase it until the first term becomes equal to one of the other terms. Hence the candidates for the optimal f are the solutions of the equations $\frac{8^n n^2}{m} = m \cdot 2^n$ and $\frac{8^n n^2}{m} = m^2$. The solution to the first one is $m_1 = 2^n n$, and to the second one $m_2 = 2^n n^{2/3}$. Plugging in we see that $f(n, m_1) = 4^n n + 4^n n + 4^n n^2 = \Theta(4^n n^2)$, and $f(n, m_2) = 4^n n^{4/3} + 4^n n^{2/3} + 4^n n^{4/3} = \Theta(4^n n^{4/3})$. So $m = m_2 = 2^n n^{2/3}$ gives the optimal solution.

To check that $f(n, m) = \Theta(4^n n^{4/3})$ is indeed optimal consider two cases. If $m < 2^n n^{2/3}$, m is “small” and the first term should dominate, and indeed $f(n, m) > \frac{8^n n^2}{m} > 4^n n^{4/3}$. If $m \geq 2^n n^{2/3}$, then m is “big” and the other two terms should dominate, and we have $f(n, m) > m \cdot 2^n + m^2 \geq 4^n n^{2/3} + 4^n n^{4/3} > 4^n n^{4/3}$. Hence $\Theta(4^n n^{4/3})$ is the (asymptotically) best answer.

4. 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) $T(n) = 7T(n/3) + n^2$

Solution:

This is a divide-and-conquer recurrence with $a = 7$, $b = 3$, $f(n) = n^2$, and $n^{\log_b a} = n^{\log_3 7}$. Since $1 < \log_3 7 < 2$, we have that $n^2 = \Omega(n^{\log_3 7 + \epsilon})$ for some constant $\epsilon > 0$, suggesting that Case 3 of the master theorem may apply. To satisfy the regularity condition, we must have that $af(n/b) \leq cf(n)$ for some constant $c < 1$, i.e. that $7(n/3)^2 \leq cn^2 \leftrightarrow 7/9 \leq c$, which is satisfied by, for example, $c = 8/9 < 1$. Thus Case 3 of the master theorem does apply and $T(n) = \Theta(f(n)) = \Theta(n^2)$.

(b) $T(n) = 4T(n/2) + n^2\sqrt{n}$

Solution:

We have $f(n) = n^2\sqrt{n} = n^{5/2}$ and $n^{\log_b a} = n^{\log_2 4} = n^2$. Since $n^{5/2} = \Omega(n^{2+1/2})$, we look at the regularity condition in case 3 of the master theorem. We have $af(n/b) = 4(n/2)^2\sqrt{n/2} = n^{5/2}/\sqrt{2} \leq cn^{5/2}$ or $1/\sqrt{2} \leq c < 1$. Case 3 applies, and we have $T(n) = \Theta(n^2\sqrt{n})$.

(c) $T(n) = T(n - 2) + 2 \log n$

Solution:

We build a recurrence table, neglecting ceiling and floor issues:

Level	Instance Size	Work in Stackframe	Number of Stackframes	Work in Level
0	n	$2 \log n$	1	$2 \log n$
1	$n - 2$	$2 \log(n - 2)$	1	$2 \log(n - 2)$
⋮	⋮	⋮	⋮	⋮
i	$n - 2i$	$2 \log(n - 2i)$	1	$2 \log(n - 2i)$
⋮	⋮	⋮	⋮	⋮
$n/2 - 1$	2	$2 \log 2 = 2$	1	$2 \log 2 = 2$

Thus the total work $T(n)$ is given by

$$T(n) = 2 \sum_{i=0}^{n/2-1} \log(n - 2i) = 2 \sum_{i=1}^{n/2} \log 2i = 2 \sum_{i=1}^{n/2} (\log 2 + \log i) = n + 2 \sum_{i=1}^{n/2} \log i$$

Claim: $\sum_{i=1}^{n/2} \log i \in \Omega(n \log n)$

Proof:

$$\sum_{i=1}^{n/2} \log i \geq \sum_{i=n/4}^{n/2} \log i \geq \sum_{i=n/4}^{n/2} \log(n/4) = (n/4 + 1) \log n - 2(n/4 + 1) \in \Omega(n \log n).$$

Claim: $\sum_{i=1}^{n/2} \log i \in O(n \log n)$

Proof: $\sum_{i=1}^{n/2} \log i \leq \sum_{i=1}^n \log n = n \log n$

Thus $\sum_{i=1}^{n/2} \log i \in \Theta(n \log n)$ and $T(n) = n + 2\Theta(n \log n) \in \Theta(n \log n)$

Note: That $T(n) \in O(n \log n)$ can be verified easily by induction. Suppose that $T(n-2) \leq c(n-2) \log(n-2)$ for some constant $c > 0, n \geq 3$. Then

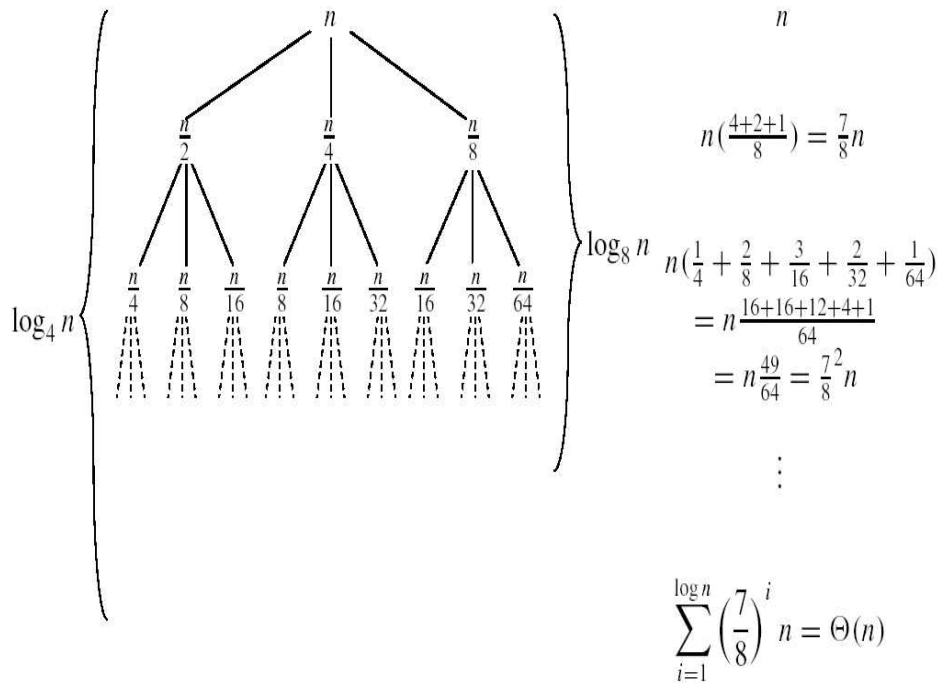
$$\begin{aligned} T(n) &= T(n-2) + 2 \log n \leq c(n-2) \log(n-2) + 2 \log n \\ &\leq (cn - 2c + 2) \log n \leq cn \log n \forall c \geq 1. \end{aligned}$$

Proving that $T(n) \in \Omega(n \log n)$ by induction is more difficult.

(d) $T(n) = T(n/2) + T(n/4) + T(n/8) + n$

Solution:

A solution can be derived from a recursion tree:



We can verify that $T(n) \in O(n)$ by induction, using the substitution method. Our inductive hypothesis is that $T(n) \leq cn$ for some constant $c > 0$. We have:

$$\begin{aligned} T(n) &= T(n/2) + T(n/4) + T(n/8) + n \\ &\leq cn/2 + cn/4 + cn/8 + n = 7cn/8 + n = (1 + 7c/8)n \leq cn \text{ if } c \geq 8. \end{aligned}$$

Therefore, $T(n) = O(n)$.

Showing that $T(n) \in \Omega(n)$ is easy: $T(n) = T(n/2) + T(n/4) + T(n/8) + n \geq n$. Since $T(n) \in O(n)$ and $T(n) \in \Omega(n)$, we have that $T(n) = \Theta(n)$.

5. Suppose that, given constants $a > 1$, $b > 1$, $c > 0$, $d \geq 0$, you have devised two recursive algorithms to solve a particular problem, with recurrence equations

$$\begin{aligned} T_1(n) &= aT_1(n/b) + cn^d \\ T_2(n) &= bT_2(n/a) + cn^d \end{aligned}$$

You wish to select the algorithm with the best asymptotic efficiency.

- (a) Under what conditions would you select Algorithm 1?
- (b) Under what conditions would you select Algorithm 2?
- (c) Under what conditions would the choice be immaterial?

Solution:

Clearly if $a = b$ the two recurrences are identical and the decision is immaterial. Now suppose that $a \neq b$, $d > \log_b a$ and $d > \log_a b$. This suggests that the solution is dominated by the “top level” or “stackframe” work for both systems, since $cn^d \in \Omega(n^{\log_b a + \epsilon})$ and $cn^d \in \Omega(n^{\log_a b + \epsilon})$ for some positive constant ϵ . To confirm this we must verify the regularity condition.

For $T_1(n)$ we must show that $ac(n/b)^d \leq \alpha cn^d$ for some positive constant $\alpha < 1$. This will hold if $a/b^d < 1 \leftrightarrow d > \log_b a$, which holds by assumption. Thus the regularity condition holds for the first system and $T_1(n) \in \Theta(n^d)$. By an analogous argument, the regularity condition must hold for $T_2(n)$, and $T_2(n) \in \Theta(n^d)$ as well. Thus if $d > \log_b a$ and $d > \log_a b$, the solutions are dominated by the top level work and the decision is again immaterial.

Now suppose that $d > \log_b a$ but $d < \log_a b$. Note that this implies that $a < b$. Then $T_1(n) \in \Theta(n^d)$, but $T_2(n) \in \Theta(n^{\log_a b})$. Since $d < \log_a b$, $T_1(n) \in o(T_2(n))$, and we should select Algorithm 1.

Now suppose that $d > \log_b a$ but $d = \log_a b$. Again, this implies that $a < b$. Again $T_1(n) \in \Theta(n^d)$, but $T_2(n) \in \Theta(n^{\log_a b \log n}) = \Theta(n^d \log n)$. Again, $T_1(n) \in o(T_2(n))$, and we should select Algorithm 1.

By symmetry, if $d > \log_a b$ but $d \leq \log_b a$ then we should select Algorithm 2.

Now suppose that $d < \log_b a$ and $d < \log_a b$, with $a \neq b$. Now both algorithms are dominated by base cases, and $T_1(n) \in \Theta(n^{\log_b a})$ and $T_2(n) \in \Theta(n^{\log_a b})$. Thus we select Algorithm 1 if $a < b$ and Algorithm 2 if $a > b$.

The decision can be summarized as follows:

IF $a = b$ OR ($d > \log_b a$ AND $d > \log_a b$) THEN

 IMMATERIAL

ELSE IF $a < b$

 SELECT ALGORITHM 1

ELSE

 SELECT ALGORITHM 2

Note: the following solution is also awarded full marks:

IF $a < b$

 SELECT ALGORITHM 1

ELSE

 SELECT ALGORITHM 2

6. You are asked to write an efficient algorithm for time-stamping an edge (a transition from one value to another) in a binary time signal. Front-end processing passes your module the signal as an n -element bit array containing exactly one edge (e.g., [00111] for $n = 5$). The edge may be of either positive or negative polarity. The temporal sampling rate is 1GHz.

- (a) Design a sublinear algorithm *findedge* that reports the delay of the edge (in nsec). Use pseudocode to describe your algorithm.

Solution:

Since we know that the signal contains exactly one edge, the problem can be solved with binary search. Here is a recursive algorithm (MATLAB code) that reports the location of the edge.

```
function y=findedge(A,p,q)
```

```
if (q-p<2)
```

```
    y=(p+q)/2;
```

```
else
```

```
    r=floor((p+q)/2);
```

```
    if A(p)~=A(r)
```

```
        y=findedge(A,p,r);
```

```
    else
```

```
        y=findedge(A,r,q);
```

end
end

Notes:

- The algorithm is called with $p = 1$ and $q = n$, where n is the length of the array A .
- The algorithm can handle edges of both positive and negative polarity.
- Since an edge is a transition between two values, it is conventional to report the location of the edge as halfway between the adjacent pixels of differing value. Thus in our example [00111], the edge is at location 2.5.
- You are asked to report the delay of the edge. It is natural to express this delay relative to the beginning of the first time sample: in our example the delay is 2.5 units. Since the sampling rate is 1GHz, the sampling period is 1 nsec, and the delay is 2.5 nsec.

(b) Write a recurrence equation for your algorithm.

Solution:

$$T(n) = T(n/2) + \Theta(1)$$

(c) Determine tight asymptotic bounds for your algorithm using

i. The method of substitution

Solution:

We rewrite the recurrence relation as $T(n) = T(n/2) + d$, where d is some positive constant. We guess (e.g. by using a recursion tree) that the solution is $T(n) \in \Theta(\log n)$. To show that $T(n) \in O(\log n)$, our inductive hypothesis is that $T(n) \leq c \log n$ for some constant $c > 0$. Then

$$T(n) = T(n/2) + d \leq c \log n/2 + d = c \log n - c + d \leq c \log n \text{ if } c \geq d.$$

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

To show that $T(n) \in \Omega(\log n)$, our inductive hypothesis is $T(n) \geq c \log n$ for some constant $c > 0$. Then

$$T(n) = T(n/2) + d \geq c \log n/2 + d = c \log n - c + d \geq c \log n \text{ if } c \leq d.$$

Thus $T(n) \in \Omega(\log n)$.

Since $T(n) \in O(\log n)$ and $T(n) \in \Omega(\log n)$, $T(n) \in \Theta(\log n)$.

ii. The recursion tree method

Solution:

Level	Instance Size	Work in Stackframe	Number of Stackframes	Work in Level
0	n	$\Theta(1)$	1	$\Theta(1)$
1	$n/2$	$\Theta(1)$	1	$\Theta(1)$
\vdots	\vdots	\vdots	\vdots	\vdots
i	$n/2^i$	$\Theta(1)$	1	$\Theta(1)$
\vdots	\vdots	\vdots	\vdots	\vdots
$\log n$	1	$\Theta(1)$	1	$\Theta(1)$

Thus the total work $T(n)$ is given by

$$T(n) = \sum_{i=0}^{\log n} \Theta(1) \in \Theta(\log n).$$

iii. The master method

Solution:

Since $a = 1$, $b = 2$ and $f(n) \in \Theta(1)$, We have $n^{\log_b a} = n^{\log_2 1} = 1 \in \Theta(f(n)) = \Theta(1)$. Thus $T(n) = \Theta(n^{\log_b a} \log n) = \Theta(\log n)$.

(d) You are now asked to write an efficient algorithm for estimating the duration of a single pulse in the binary time signal (e.g., [00110] for $n = 5$). Show that, in general, no sublinear algorithm exists for this problem.

Solution:

Note that the pulse cannot include the first and last samples, otherwise it would be an edge.

- Claim: A correct algorithm must examine at least $\lfloor \frac{n-2}{2} \rfloor$ array locations.
- Proof: Consider an algorithm that examines less than $\lfloor \frac{n-2}{2} \rfloor$ array locations.
 - Claim: There must be at least two adjacent locations that are unexamined.
 - Proof: Suppose there are no two adjacent locations that are unexamined. This means that there must be at least one examined location in each successive pair of the central $n-2$ samples. Thus the number of examined locations $m \geq \lfloor \frac{n-2}{2} \rfloor$. This contradicts our assertion that $m < \lfloor \frac{n-2}{2} \rfloor$. Hence there must be at least two adjacent unexamined locations.

To determine a test case on which the algorithm fails, we first give the algorithm the (invalid) input consisting entirely of zeros, observe the locations examined, and determine two adjacent unexamined locations $(j, j+1)$. We then provide two test cases: one in which all bits are 0 except bit j and another for which all bits are 0 except bits j and $j+1$. Since the algorithm will not examine either locations j or $j+1$ it will not be able to distinguish these signals. Hence the algorithm must be incorrect.

Thus a correct algorithm must examine at least $\lfloor \frac{n-2}{2} \rfloor$ array locations, and hence must have running time $T(n) \in \Theta(n)$.

- (e) Suppose that front-end processing guarantees the pulse to be at least n/a nsec in duration, where $a > 1$ is a constant. Design an algorithm that uses your *findedge* routine to solve the problem in sublinear time. What is the asymptotic running time of your algorithm?

Solution:

Here is an algorithm, again in MATLAB code, that uses *findedge* to compute the pulse duration.

```
function y=findpulse(A,n,a)

r=2;
dr=ceil(n/a);
while (r<n) & (A(r)==A(1))
    r=r+dr;
end

y=findedge(A,r,n)-findedge(A,1,r);
```

The loop samples the signal at regular intervals of n/a , just finely enough to ensure that one will land on the pulse. Note that the maximum number of samples m is a constant: $m = \frac{n-3}{\lceil n/a \rceil} + 1 \approx a + 1 \in \Theta(1)$. Thus

$$T(n) = \frac{n-3}{\lceil n/a \rceil} + 1 + \Theta(\log(n - r + 1)) + \Theta(\log r) \in \Theta(\log n).$$