

Question 1

Below are two different ways to prove that, for all integers $r > 0$, $\sum_{i=1}^n i^r$ is $\Theta(n^{r+1})$.

Direct Proof.

We first show that $\sum_{i=1}^n i^r$ is $O(n^{r+1})$:

$$\begin{aligned} \sum_{i=1}^n i^r &= 1^r + 2^r + \dots + n^r \\ &\leq n(n^r) \text{ because there are } n \text{ terms and } n^r \text{ is the largest} \\ &= n^{r+1}. \end{aligned}$$

Next we show that $\sum_{i=1}^n i^r$ is $\Omega(n^{r+1})$:

$$\begin{aligned} \sum_{i=1}^n i^r &= 1^r + 2^r + \dots + n^r \\ &= 1^r + 2^r + \dots + \left\lfloor \frac{n}{2} \right\rfloor^r + \left(\left\lfloor \frac{n}{2} \right\rfloor + 1 \right)^r + \dots + n^r \\ &\geq \left\lfloor \frac{n}{2} \right\rfloor^r + \left(\left\lfloor \frac{n}{2} \right\rfloor + 1 \right)^r + \dots + n^r \\ &\geq \left(\left\lfloor \frac{n}{2} \right\rfloor + 1 \right) \left\lfloor \frac{n}{2} \right\rfloor^r \text{ because there are } \left\lfloor \frac{n}{2} \right\rfloor + 1 \text{ terms and } \left\lfloor \frac{n}{2} \right\rfloor^r \text{ is the smallest} \\ &\geq \left\lfloor \frac{n}{2} \right\rfloor^{r+1} \text{ because } \left\lfloor \frac{n}{2} \right\rfloor + 1 \geq \left\lfloor \frac{n}{2} \right\rfloor \\ &\geq \left(\frac{n}{2} \right)^{r+1} \text{ because } \left\lfloor \frac{n}{2} \right\rfloor \geq \frac{n}{2} \\ &\geq \left(\frac{1}{2^{r+1}} \right) n^{r+1} \end{aligned}$$

Induction Proof.

We first show that $\sum_{i=1}^n i^r$ is $O(n^{r+1})$; that is, that there exist constants $c > 0$ and $n_0 \geq 1$ such that $\sum_{i=1}^n i^r \leq cn^{r+1}$ for all $n \geq n_0$. We prove this for $n_0 = 1$:

base step: When $n = 1$ we have $\sum_{i=1}^n i^r = 1$ and $n^{r+1} = 1$ so $\sum_{i=1}^n i^r \leq cn^{r+1}$ provided that $c \geq 1$.

induction step: Assume that $\sum_{i=1}^k i^r \leq ck^{r+1}$ for some $k \geq 1$.

$$\begin{aligned} \sum_{i=1}^{k+1} i^r &= 1^r + 2^r + \dots + k^r + (k+1)^r \\ &\leq ck^{r+1} + (k+1)^r \text{ because of our induction assumption} \\ &\leq c(k+1)^r k + (k+1)^r \text{ because } k^{r+1} \leq k^r k \leq (k+1)^r k \\ &= c(k+1)^r(k+1) \\ &= c(k+1)^{r+1} \end{aligned}$$

Next we show that $\sum_{i=1}^n i^r$ is $\Omega(n^{r+1})$; that is, that there exist constants $c > 0$ and $n_0 \geq 1$ such that $n^{r+1} \leq c \sum_{i=1}^n i^r$ for all $n \geq n_0$. For this proof we use a specific form of the binomial theorem:

$$(a+1)^m = \sum_{i=0}^m \binom{m}{i} a^i$$

where $\binom{m}{i} = \frac{m!}{i!(m-i)!}$. Once again we prove this for $n_0 = 1$:

base step: Same as for the big-O proof above.

induction step: Assume that $k^{r+1} \leq c \sum_{i=1}^k i^r$ for some $k \geq 1$.

$$\begin{aligned}
 c \sum_{i=1}^{k+1} i^r &= c(1^r + 2^r + \dots + k^r + (k+1)^r) \\
 &= c(1^r + 2^r + \dots + k^r) + c(k+1)^r \\
 &\geq k^{r+1} + c(k+1)^r \text{ because of our induction assumption} \\
 &= k^{r+1} + c \sum_{j=0}^r \binom{r}{j} k^j \text{ by the binomial theorem} \\
 &= k^{r+1} + c \binom{r}{r} k^r + c \binom{r}{r-1} k^{r-1} + c \binom{r}{r-2} k^{r-2} + \dots + c \binom{r}{0} \\
 &\geq \binom{r+1}{r+1} k^{r+1} + \binom{r+1}{r} k^r + \binom{r+1}{r-1} k^{r-1} + \binom{r+1}{r-2} k^{r-2} + \dots + \binom{r+1}{0} \\
 &\quad \text{whenever } c \binom{r}{i} k^i \geq \binom{r+1}{i} k^i \text{ for all } 0 \leq i \leq r \\
 &= (k+1)^{r+1} \text{ by the binomial theorem}
 \end{aligned}$$

We still have to determine when $c \binom{r}{i} \geq \binom{r+1}{i}$:

$$\begin{aligned}
 \binom{r+1}{i} &= \frac{(r+1)!}{i!(r+1-i)!} \\
 &= \frac{(r+1)r!}{i!(r+1-i)(r-i)!} \\
 &= \binom{r+1}{r+1-i} \frac{r!}{i!(r-i)!} \\
 &= \frac{r+1}{r+1-i} \binom{r}{i} \\
 &\leq \frac{r+1}{r+1-r} \binom{r}{i} \\
 &= (r+1) \binom{r}{i}.
 \end{aligned}$$

In other words, c must be at least $r+1$.

Question 2

The algorithm is simple: read the input expression from left to right while evaluating as much of the expression that we have seen as possible. As an example, consider evaluating the expression $(1 \& \sim (0 | 1 \wedge \sim 0 | 1) \& 1)$. To compare the precedence of operators we define a precedence function as follows: $\text{precedence}(\sim) > \text{precedence}(\&) > \text{precedence}(\wedge) > \text{precedence}(|) > \text{precedence}(())$.

```
( 1 & ~ ( 0 | 1 ^ ~ 0 | 1 ) & 1 )
(
( 1
( 1 &
( 1 & ~
( 1 & ~ (
( 1 & ~ ( 0
( 1 & ~ ( 0 |
( 1 & ~ ( 0 | 1
( 1 & ~ ( 0 | 1 ^
( 1 & ~ ( 0 | 1 ^ ~
( 1 & ~ ( 0 | 1 ^ ~ 0
( 1 & ~ ( 0 | 1 ^ ~ 0 |
( 1 & ~ ( 0 | 1 ^ 1 |
  because precedence('|') < precedence('~') and ~ 0 = 1
( 1 & ~ ( 0 | 0 |
  because precedence('|') < precedence('^') and 1^1 = 0
( 1 & ~ ( 0 |
  because precedence('|') = precedence('|') and 0|0 = 0
( 1 & ~ ( 0 | 1
( 1 & ~ ( 0 | 1 )
( 1 & ~ ( 1 )
  because we have ')' and 0|1 = 1
( 1 & ~ 1
  because (1) = 1
( 1 & ~ 1 &
( 1 & 0 &
  because precedence('&') < precedence('~') and ~ 1 = 0
( 1 & 0 & 1
( 1 & 0 & 1 )
( 1 & 0 )
  because we have ')' and 0&1 = 0
( 0 )
  because we have ')' and 1&0 = 0
  0
  because (0) = 0
```

In this example, notice that we modify only the rightmost symbols in each row to obtain the next row of symbols. Because a stack can be used to efficiently store a sequence of symbols and modify the symbols at one end of the sequence, the algorithm we used in the example above can be efficiently implemented using stacks. The algorithm reads through the expression from left

to right pushing the symbols it reads onto a stack and evaluating sub-expressions at the top of the stack whenever possible.

ALGORITHM evaluate(e)

Input: e is a valid expression on bits '0' and '1' and bitwise operators '&', '^', '|', '~', '(' and ')'.
Output: 0 if $e = 0$ and 1 otherwise.

$s \leftarrow$ empty stack

```
for each character  $c$  in  $e$  (from left to right) do
  if  $c$  is equal to one of '0', '1' or '(' or is a unary operator then
    s.push( $c$ )
  else if  $c$  is equal to ')' then
    evaluateStack( $s$ , precedence('(') + 1)
     $b \leftarrow$  s.pop()
    s.pop()
    s.push( $b$ )
  else
    evaluateStack( $s$ , precedence( $c$ ))
    s.push( $c$ )
evaluateStack( $s$ , precedence('('))
return s.top()
```

ALGORITHM evaluateStack(s, p)

Input: s is a stack containing bits and operators and p is an integer. Furthermore, if s is equal to (s_1, s_2, \dots, s_n) where s_1 is the top element in the stack then the expression created by removing each '(' operator from $s_n s_{n-1} \dots s_1$ is valid.

Output: If s_i is the operator nearest the top of the stack with $\text{precedence}(s_i) < p$ then the algorithm returns with s equal to $(t, s_i, s_{i+1}, \dots, s_n)$ where t is the result of evaluating the expression $s_{i-1} s_{i-2} \dots s_1$. Otherwise, the algorithm returns with s equal to (t) where t is the result of evaluating the expression $s_n s_{n-1} \dots s_1$.

```
while s.size() > 1 do
   $b \leftarrow$  s.pop() //the top element is a bit
  if precedence(s.top()) < p then
    s.push( $b$ )
  return
if o is unary then
  s.push(s.pop()  $b$ ) //e.g. pushes  $\sim 0 = 1$  if  $b = 0$  and s.pop() returns  $\sim$ 
else
  s.push( $b$  s.pop() s.pop()) //e.g. pushes  $1 \& 0 = 0$  if  $b = 1$  and s.pop() returns  $\&$  and 0
```