MAT1830

Lecture 27: Recursive Algorithms

Recursion may be used to define functions whose definition normally involves "$\cdots$", to give algorithms for computing these functions, and to prove some of their properties.

Let a and b be integers with $a \leq b$ and let $t : \{a, a+1, \ldots, b\} \to \mathbb{R}$ be a function.

We define $\sum_{i=a}^{b} t(i)$ to be $t(a) + t(a+1) + \cdots + t(b)$.

Examples

$$\sum_{i=7}^{9} \frac{1}{i} = \frac{1}{7} + \frac{1}{8} + \frac{1}{9}$$

$$\sum_{i=3}^{7} (-1)^i 2i = -6 + 8 - 10 + 12 - 14$$

$$5^2 + 7^2 + 9^2 + 11^2 = \sum_{i=2}^{5} (2i+1)^2$$

27.1 Sums

Example. $1 + 2 + 3 + \cdots + n$

This is the function $f(n)$ defined by

Initial value. $f(1) = 1$
Recurrence relation. $f(k+1) = f(k) + (k+1)$

$$1 + 2 + 3 + \cdots + n = \sum_{i=1}^{n} i$$

Example. $1 + a + a^2 + \cdots + a^n$

This is the function $g(n)$ defined by

Initial value. $g(0) = 1$
Recurrence relation. $g(k+1) = g(k) + a^{k+1}$

$$1 + a + a^2 + \cdots + a^n = \sum_{i=0}^{n} a^i$$

We can use this relation to prove by induction that $g(n) = \frac{a^{n+1}-1}{a-1}$ (a formula for the sum of a geometric series), provided $a \neq 1$.

Proof

Base step. For $n = 0$, $1 = g(0) = \frac{a^{0+1}-1}{a-1}$, as required.

Induction step. We want to prove

$$g(k) = \frac{a^{k+1}-1}{a-1} \Rightarrow g(k+1) = \frac{a^{k+2}-1}{a-1}.$$

Well,

$$\begin{aligned} g(k) &= \frac{a^{k+1}-1}{a-1} \\ \Rightarrow g(k+1) &= \frac{a^{k+1}-1}{a-1} + a^{k+1} \\ \Rightarrow g(k+1) &= \frac{a^{k+1}-1+(a-1)a^{k+1}}{a-1} \\ &= \frac{a^{k+2}+a^{k+1}-a^{k+1}-1}{a-1} \\ &= \frac{a^{k+2}-1}{a-1} \text{ as required.} \end{aligned}$$

This completes the induction.

Example Let $a \neq 1$ be a real number. Prove that $\sum_{i=0}^{n} a^i = \frac{a^{n+1}-1}{a-1}$ for all integers $n \geq 0$.

Solution Let $P(n)$ be the statement "$\sum_{i=0}^{n} a^i = \frac{a^{n+1}-1}{a-1}$".

Base step. $\sum_{i=0}^{0} a^i = a^0 = 1$ and $\frac{a^{0+1}-1}{a-1} = \frac{a-1}{a-1} = 1$, so $P(0)$ is true.

Induction step. Suppose that $P(k)$ is true for some integer $k \geq 0$. This means that $\sum_{i=0}^{k} a^i = \frac{a^{k+1}-1}{a-1}$.

We want to prove that $P(k+1)$ is true. We want to show that $\sum_{i=0}^{k+1} a^i = \frac{a^{k+2}-1}{a-1}$.

$$\begin{aligned}
\sum_{i=0}^{k+1} a^i &= \sum_{i=0}^{k} a^i + a^{k+1} \\
&= \frac{a^{k+1}-1}{a-1} + a^{k+1} && \text{(by } P(k)\text{)} \\
&= \frac{a^{k+1}-1}{a-1} + \frac{a^{k+2}-a^{k+1}}{a-1} \\
&= \frac{a^{k+2}-1}{a-1}
\end{aligned}$$

So $P(k+1)$ is true.

This proves that $P(n)$ is true for each integer $n \geq 0$.

Flux Exercise

The value of $\sum_{i=0}^{3}(-1)^{i+1}\, i^3$ is

A: -36

B: -20

C: 20

D: 36

Answer

$$\begin{aligned}\sum_{i=0}^{3}(-1)^{i+1}\, i^3 &= (-1)^1 \times 0^3 + (-1)^2 \times 1^3 + (-1)^3 \times 2^3 + (-1)^4 \times 3^3 \\ &= -0 + 1 - 8 + 27 \\ &= 20\end{aligned}$$

So C.

Questions

27.1 Rewrite the following sums using $\sum$ notation.

- $1 + 4 + 9 + 16 + \cdots + n^2$
- $1 - 2 + 3 - 4 + \cdots - 2n$

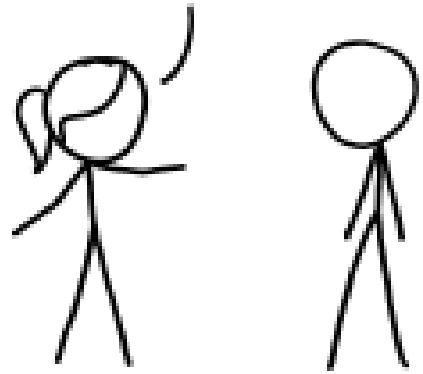
Answer The first sum can be written as

$$\sum_{i=1}^{n} i^2.$$

The second sum can be written as

$$\sum_{i=1}^{2n} (-1)^{i+1} i \quad \text{or} \quad \sum_{i=1}^{2n} (-1)^{i-1} i.$$

I MET A TRAVELER FROM AN ANTIQUE LAND WHO SAID: "I MET A TRAVELER FROM AN ANTIQUE LAND, WHO SAID: "I MET A TRAVELER FROM AN ANTIQUE LAND, WHO SAID: "I MET...



Let a and b be integers with $a \leq b$ and let $t : \{a, a+1, \ldots, b\} \to \mathbb{R}$ be a function.

We define $\prod_{i=a}^{b} t(i)$ to be $t(a) \times t(a+1) \times \cdots \times t(b)$.

Examples

$$\prod_{i=7}^{9}(x-i) = (x-7)(x-8)(x-9)$$

$$\prod_{i=2}^{5}\binom{6}{i} = \binom{6}{2} \times \binom{6}{3} \times \binom{6}{4} \times \binom{6}{5}$$

$$\frac{3}{16} \times \frac{4}{25} \times \frac{5}{36} \times \cdots \times \frac{9}{100} = \prod_{i=3}^{9} \frac{i}{(i+1)^2}$$

27.2 Products

Example. $1 \times 2 \times 3 \times \cdots \times n$

This is the function $n!$ defined recursively by

Initial value. $0! = 1$
Recurrence relation. $(k+1)! = (k+1) \times k!$

$$n! = \prod_{i=1}^{n} i$$

27.3 Sum and product Notation

$$1 + 2 + 3 + \cdots + n \text{ is written } \sum_{k=1}^{n} k,$$

$$1 + a + a^2 + \cdots + a^n \text{ is written } \sum_{k=0}^{n} a^k.$$

Σ is capital sigma, standing for "sum."

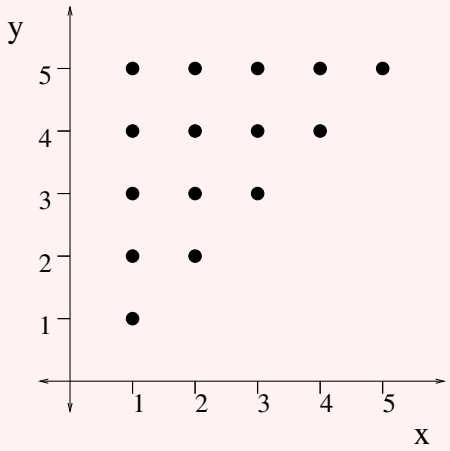
$$1 \times 2 \times 3 \times \cdots \times n \text{ is written } \prod_{k=1}^{n} k.$$

Π is capital pi, standing for "product."

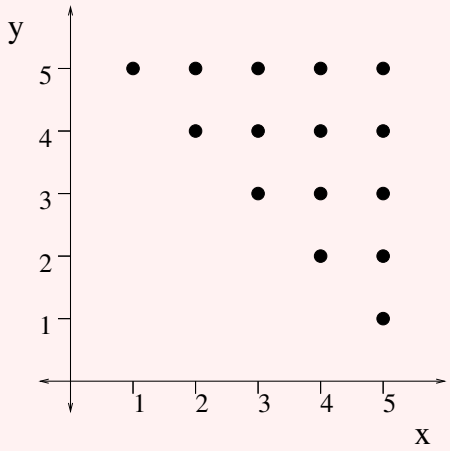
Flux Exercise

$\sum_{x=1}^{5} \sum_{y=x}^{5} f(x, y)$ adds the values of f at which points?

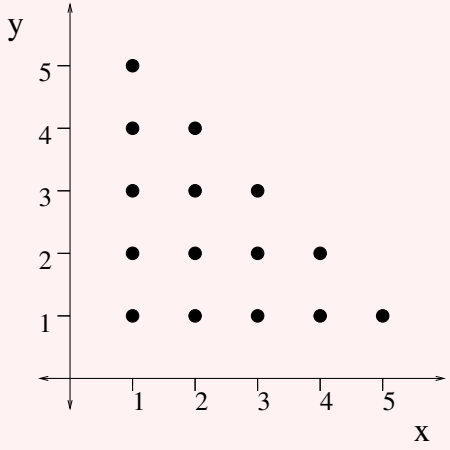
A.



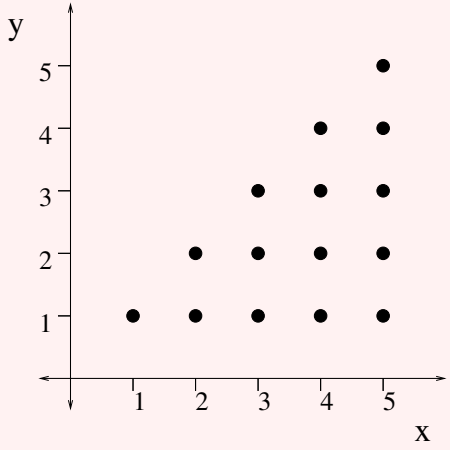
C.



B.



D.



Answer A.

27.4 Binary search algorithm

Given a list of n numbers in order

$$x_1 < x_2 < \cdots < x_n,$$

we can find whether a given number a is in the list by repeatedly "halving" the list.

The algorithm `binary search` is specified recursively by a *base step* and a *recursive step*.

Base step. If the list is empty,
report 'a is not in the list.'

Recursive step If the list is not empty, see whether its middle element is a. If so, report 'a found.'
Otherwise, if the middle element $m > a$, `binary search` the list of elements $< m$. And if the middle element $m < a$, `binary search` the list of elements $> m$.

Note: If the list is of even length the algorithm chooses one of the two middle elements.

Example Performing binary search for 42.

13	15	16	19	22	27	28	30	36	40	45	47	51	70	77	79	82
13	15	16	19	22	27	28	30	<u>36</u>	40	45	47	51	70	77	79	82
									40	45	47	51	70	77	79	82
									40	45	47	<u>51</u>	70	77	79	82
									40	45	47					
									40	<u>45</u>	47					
									40							
									<u>40</u>							

output: 42 is not in the list

27.5 Correctness

We prove that the algorithm works on a list of n items by strong induction on n.

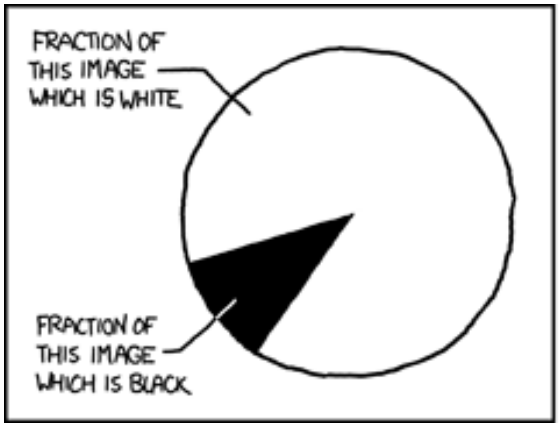
Base step. The algorithm works correctly on a list of 0 numbers, by reporting that a is not in the list.

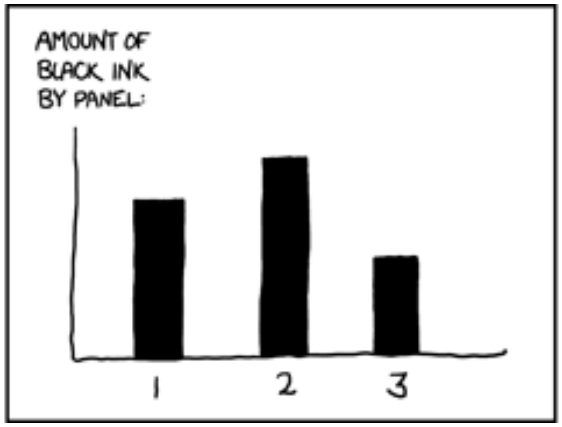
Induction step. Assuming the algorithm works correctly on any list of $< k+1$ numbers, suppose we have a list of $k+1$ numbers.

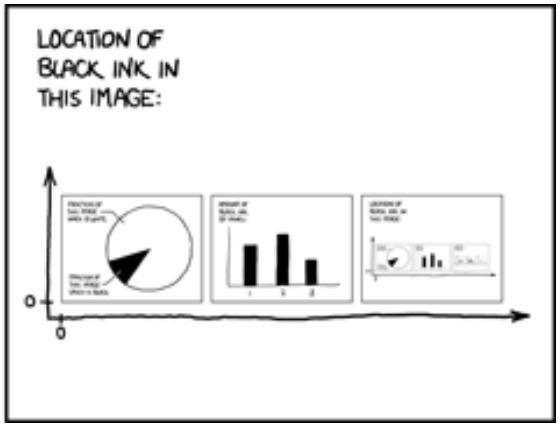
The recursive step either finds a as the middle number in the list, or else produces a list of $< k+1$ numbers to search, which by assumption it will do correctly.

This completes the induction.

Remark. This example shows how easy it is to prove correctness of recursive algorithms, which may be why they are popular despite the practical difficulties in implementing them.


FRACTION OF THIS IMAGE WHICH IS WHITE
FRACTION OF THIS IMAGE WHICH IS BLACK


AMOUNT OF BLACK INK BY PANEL:
1
2
3


LOCATION OF BLACK INK IN THIS IMAGE:
0
0

27.6 Running time

$\log_2 n$ is the number x such that

$$n = 2^x.$$

For example, $1024 = 2^{10}$, and therefore

$$\log_2 1024 = 10.$$

Similarly $\log_2 512 = 9$, and hence $\log_2 1000$ is between 9 and 10.

Repeatedly dividing 1000 by 2 (and discarding remainders of 1) runs for 9 steps:

$$500, 250, 125, 62, 31, 15, 7, 3, 1$$

The 10 halving steps for 1024 are

$$512, 256, 128, 64, 32, 16, 8, 4, 2, 1$$

This means that the binary search algorithm would do at most 9 "halvings" in searching a list of 1000 numbers and at most 10 "halvings" for 1024 numbers.

More generally, binary search needs at most $\lfloor \log_2 n \rfloor$ "halvings" to search a list of n numbers, where $\lfloor \log_2 n \rfloor$ is the *floor* of $\log_2 n$, the greatest integer $\leqslant \log_2 n$.

Remark. In an alphabetical list of 1,000,000 names, which is just under 2^{20}, it takes at most 19 halvings (using alphabetical order) to find whether a particular name is present.

Example Prove that `binary search` produces the correct result in at most n halvings on any list of length less than 2^{n+1} for all integers $n \geq 0$.

Solution Let $P(n)$ be the statement "`binary search` works in at most n halvings on any list of length less than 2^{n+1}".

Base step. If the list has less than $2^1 = 2$ elements then it just has 1 element and we only need to check that. We do not need any halvings. So $P(0)$ is true.

Induction step. Suppose that $P(k)$ is true for some integer $k \geq 0$. This means that `binary search` works in at most k halvings on any list of length less than 2^{k+1}

We want to prove that $P(k+1)$ is true. We want to show that `binary search` works in at most $k+1$ halvings on any list of length less than 2^{k+2}.

If we begin with a list of length less that 2^{k+2} and choose a middle element m, then the list of elements $< m$ has length less than 2^{k+1} and the list of elements $> m$ has length less than 2^{k+1}.
(Otherwise the original list would have length at least $1 + 2^{k+1} + (2^{k+1} - 1) = 2^{k+2}$.)

If $m = a$, we finish in 0 halvings.

If $m \neq a$, then we run `binary search` on a list of length less than 2^{k+1}.
By $P(k)$ this produces the correct result in at most k halvings.
So overall we get the correct result in at most $k+1$ halvings. So $P(k+1)$ is true.

This proves that $P(n)$ is true for each integer $n \geq 0$.

27.7 20 questions

A mathematically ideal way to play 20 questions would be to divide the number of possibilities in half with each question.

E.g. if the answer is an integer, do binary search on the list of possible answers. If the answer is a word, do binary search on the list of possible answers (ordered alphabetically). If this is done, then 20 questions suffice to find the correct answer out of $2^{20} = 1,048,576$ possibilities.

Question

27.3 Imagine a game where the object is to identify a natural number between 1 and 2^{20} using 20 questions with YES-NO answers. The lecture explains why 20 questions are sufficient to identify any such number.
Explain why less than 20 YES-NO questions are not always sufficient.

Answer

For any sequence of 19 YES-NO questions we could write next to each of the numbers the string of 19 answers we would give if that was our number.

For example, maybe we would write YYNY$\cdots$NN next to 12 and so on.

There are 2^{19} possible strings of Ys and Ns and there are 2^{20} numbers.

By the pigeonhole principle there will be (at least) two numbers that get the same string.

So for any possible sequence of questions, there will be (at least) two numbers they cannot distinguish between.