VICTORIA UNIVERSITY OF WELLINGTON

Te Whare Wananga o te Upoko o te Ika a Maui

EXAMINATIONS — 2003

MID-YEAR

COMP 303

Design and Analysis of Algorithms

Time Allowed: 3 Hours

Instructions: There are 6 questions of various weights.
Answer all questions.
Available marks total 180, so allow about one minute per mark.

Calculators **are** permitted.
Foreign language dictionaries **are** permitted.

Useful results

Asymptotic notation

$$\begin{aligned} O(f(n)) &= \{g(n) \mid (\exists c)(\mathbf{aa}\, n)[g(n) \leq c.f(n)]\} \\ \Omega(f(n)) &= \{g(n) \mid (\exists c)(\mathbf{aa}\, n)[g(n) \geq c.f(n)]\} \\ \Theta(f(n)) &= O(f(n)) \cap \Omega(f(n)) \end{aligned}$$

"**aa**" means "almost all"

Master theorem

Let $T(n)$ be defined by the recurrence $T(n) = aT(n/b) + f(n)$.

1. If $f(n) \in O(n^{\log_b a - \epsilon})$ for some constant $\epsilon > 0$ then $T(n) \in \Theta(n^{\log_b a})$.
2. If $f(n) \in \Theta(n^{\log_b a})$ then $T(n) \in \Theta(n^{\log_b a} \log_2 n)$.
3. If $f(n) \in \Omega(n^{\log_b a + \epsilon})$ for some constant $\epsilon > 0$, and if $af(n/b) \leq cf(n)$ for some constant $c < 1$ and all sufficiently large n then $T(n) \in \Theta(f(n))$.

Question 1. [31 marks]

(a) [3 marks] What do we mean when we say that an algorithm runs in time:

(i) $O(n)$,
Answer:
The algorithms running time is bounded above by some linear multiple of a barometer measure n

(ii) $\Omega(n)$,
Answer:
The algorithms running time is bounded below by some linear multiple of a barometer measure n

(iii) $\Theta(n)$
Answer:
The algorithms running time has an exact asymptotic bound t, where $dn \leq t \leq cn$ for constants d, n and n is a barometer measure

(b) [5 marks] What is the *principle of invariance*? Give two examples of why it applies to computer programs and say how it justifies our use of asymptotic notation in the analysis of algorithms.
Answer:
The principle of invariance says that
"Two different reasonable implementations of an algorithm will differ by at most a constant multiplicative factor."
It applies to computer programs, since i) CPU speeds may vary, ii) The quality of the compiler may vary.
It justifies our use of asymptotic notation in the analysis of algorithms because of the constant factor which appears in the definition of $O(n)$, $\Theta(n)$ and $\Omega(n)$.

(c) [3 marks] What is the asymptotic running time of the following algorithm (ie. $\Theta(f(n))$, where n is some appropriate measure of the size of the input):

Algorithm - *Sum of squares*
input - `lst` A list containing the input numbers
output - `sum` the sum of the squares of the input list.

```
sum ← 0
while (size(lst)>0) repeat
    f ← first(lst)          // put the first element of lst in f
    sum ← $f^2$+sum          // square it and add to sum
    lst ← rest(lst)         // remove the first element from lst
return(sum)
```

continued...

Answer:
The asymptotic running time of the above algorithm is $\Theta(n)$, where n is the length of the list `lst`

(d) [5 marks]

(i) The expression $n^2 + n$ is in which of these asymptotic sets:
$O(n),\ O(n \log_2 n),\ \Omega(n \log_2 n)$
Answer:
$\Omega(n \log_2 n)$

(ii) The expression $\log_2 n$ is in which of these asymptotic sets:
$\Omega(n),\ \Omega(1),\ O(n)$
Answer:
$\Omega(1),\ O(n)$

(iii) Using the definitions for O, Ω, and Θ given on the front page of this paper, show that

$$5n^2 + 200 \in \Theta(n^2)$$

Answer:
proof:

1) prove that $5n^2 + 200 \in O(n^2)$
Let $c = 6$, now $\forall n > 20$

$$\begin{aligned} 200 &< n^2 \\ 5n^2 + 200 &< 5n^2 + n^2 \\ 5n^2 + 200 &< 6n^2 = cn^2 \end{aligned}$$

therefore **aa** n $5n^2 + 200 < cn^2$ so $5n^2 + 200 \in O(n^2)$

2) prove that $5n^2 + 200 \in \Omega(n^2)$
Let $c = 4$, $\forall n > 15n^2 + 100 > 4n^2$
therefore **aa** n $5n^2 + 200 < cn^2$ so $5n^2 + 200 \in \Omega(n^2)$

therefore

$$5n^2 + 200 \in O(n^2) \cap \Omega(n) = \Theta(n^2)$$

(e) [15 marks] For each of the following recurrence relations, give the asymptotic complexity (Θ bound) of $T(n)$. **Justify your answers**, using the Master Theorem (provided on the front page of this paper) or mathematical induction, as appropriate.

(i) $T(n) = \begin{cases} 1, & \text{if } n = 0 \\ 25T(\frac{n}{5}) + n \log_2 n, & \text{otherwise} \end{cases}$
Answer:
$f(n) = n \log_2 n \in O(n^{2-0.1})$, since $\log_5 25 = 2$ then the first case of the Master theorem holds and the solution is

$$T(n) = \Theta(n^2)$$

continued...

(ii) $T(n) = \begin{cases} 1, & \text{if } n = 0 \\ 5T(\frac{n}{25}) + n^2, & \text{otherwise} \end{cases}$

Answer:
$f(n) = n^2 \in \Omega(n^{log_{25}5+\epsilon})$ for $\epsilon = \frac{1}{2}$ The regularity condition must also hold, $a \cdot f(n/b) = 5\frac{n^2}{625} < cn^2$ for $c = 1$, therefore the 3rd case of the master theorem holds and the solution is

$$T(n) \in \Theta(n^2)$$

(iii) $T(n) = \begin{cases} 1, & \text{if } n = 0 \\ 2T(n-1), & \text{otherwise} \end{cases}$

Answer:

$$T(n) = 2^n$$

Proof by induction
basis for induction: $T(0) = 2^0 = 1$
assume result is true for $m < n$ then $T(n-1) = 2^{n-1}$
now

$$\begin{aligned} T(n) &= 2T(n-1) \\ &= 2 \cdot T^{n-1} = 2^n \end{aligned}$$

continued...

Question 2. [20 marks]

A computer printer has a number of documents to print $d_1, d_2, \cdots, d_n$. The documents have $p_1, p_2, \cdots p_n$ pages respectively. The quality of the printing decreases as the ink cartridge is used up. This means that the later jobs are printed with a lower quality. In order to keep the maximum number of users happy, it is desirable to finish the maximum number of jobs earlier before the quality of the printing degrades. To achieve this we keep track of the number of pages printed (the printer page count) and assign a *fade level*, f_i to each document. This is the printer page count on the last page of the document.

So the fade level for the first document (f_1) is equal to the number of pages in that document, the fade level for the next document (f_2) is the sum of f_1 and the number of pages in the second document, etc.

The goal is to order the documents to minimize the average fade level over all the documents. For example, suppose there are two documents, d_1 and d_2 with $p_1 = 20$ pages and $p_2 = 16$ pages. If we print d_1 first, then $f_1 = 20$ and $f_2 = 36$, so the average fade level is $(20 + 36)/2 = 28$. If we print d_2 first, then $f_1 = 16$ and $f_2 = 36$, so the average fade level is $(16 + 36)/2 = 26$.

(a) [6 marks] Describe an efficient **greedy algorithm** that will solve this problem, returning the printing order which will minimize the average fade level.

Answer:

```
*PrintOrder*
**input:** $D = d_1, \cdots, d_n$ an indexed list of documents.
**output:** A printer stream
put all the jobs onto a priority queue (shortest jobs has highest priority).
while not(priority queue is empty) repeat
    $d_s \leftarrow$ job on top of priority queue
    remove $d_s$ from priority queue
    output $d_s$ on the printer stream
```

An alternative algorithm could order the documents first using a standard sort, then just read the documents off in that order.

(b) [8 marks] Prove that your algorithm always minimizes the average fade level. (Hint: Your algorithm is a greedy algorithm, there is a general pattern for proving the correctness of greedy algorithms.)

Answer:

Greedy choice: choose the next shortest printer job. In the algorithm given, this is the next job on the priority queue.

Acceptable solution: The final solution will minimise the fade level. This will happen as long as choices are always feasable.

continued...

Feasable partial solution: Feasability criteria : "Partial solutions minimise the fade level *so far*". This requirement follows from the greedy initial choice and the optimal substructure property, since the smallest remaining at any one time is added.

Greedy initial choice: The smallest job must come first in order to minimise the average, since the first it will give a linear contribution to every fade level, so a bigger initial value would give a bigger contribution to every fade level. Thus giving a bigger average.

Optimal substructure: We may prove optimal substructure by induction, on the document order. We shall prove that the optimal substructure follows from the documents being taken in order of size, shortest first:

1) The basis for induction follows from the greedy intial choice.
2) assume that the average fade level is minimal once $s-1$ documents have been printed i.e.$\frac{1}{s-1}\sum_{i=1}^{s-1} f_i$ is minimal,
we must prove that the average fade level is minimal once s documents have been printed i.e. $\frac{1}{s}\sum_{i=1}^{s} f_i$ is minimal.
Now $\frac{1}{s}\sum_{i=1}^{s} f_i = \frac{1}{n}\left(\sum_{i=1}^{s-1} f_i + \sum_{i=1}^{s-1} f_i + p_s\right)$ since by the greedy choice p_s is the smallest remaining and the inductive hypothesis then $\frac{1}{s}\sum_{i=1}^{s} f_i$ is minimal, therefore by induction $\frac{1}{n}\sum_{i=1}^{n} f_i$ is minimal, i.e. the fade level is minimal.

(c) [6 marks] What is the complexity of the algorithm you give in **(a)**. Specify any assumptions you make including any necessary details about your algorithm.
Answer:
The complexity of the algorithm in (a) is $\Theta(n \log n)$. The priority queue is stored on a heap, so construction is $O(n \log n)$ and removal is $O(1)$.

continued...

Question 3. [28 marks]

(a) [6 marks] Using pseudocode, define the basic structure of a typical divide-and-conquer algorithm. Explain the components of your algorithm.

Answer:
A typical divide-and-conquer algorithm

```
DivCon(P)
  if term?(P) then // terminating condition on P
    terminating case
    $P_i$← split(P) // split problem P into a number of subproblems $P_i$
    for each $i$ do
      $S_i$← DivCon($P_i$)
    Combine($S_i$) // combine sub-solutions $S_i$
```

(b) [4 marks] If a problem can be split into a number of subproblems, we can base a Divide and Conquer algorithm on these subproblems. Give some properties of the subproblems where Divide and Conquer (without memoizing) is a bad algorithmic strategy.

Answer:
When the subproblems are not small (especially when they are bigger than the problem - non-termination!!)
when there are many subproblems
when the the same subproblem is repeated.

(c) [8 marks] Outline the structure of a proof of correctness of a typical divide-and-conquer algorithm. In your answer, you should refer to *assumptions* **A** and *requirements* **R** for the subproblems.
Answer:

Direct:
assumption: $\mathbf{A(x)}$ satisfies some terminating condition. e.g. $x < n_0$ for some small n_0
requirements: $\mathbf{R(x)}$ may be proved directly
for an inductive proof this is the basis for induction

Divide:
Show that the divide step reduces the problem size and establishes the correct preconditions for the subproblems

Combine:
We may assume $\mathbf{R(\tilde{x})}$ and $\mathbf{A(x)}$, where $\mathbf{\tilde{x} < x}$, and we must establish $\mathbf{R(x)}$

(d) [6 marks] Give a *recurrence relation* that describes the running time of a typical divide-and-conquer algorithm. Explain how each part of your recurrence relates to the components of your divide-and-conquer algorithm in **(a)**.

Answer:

$$T(n) = \begin{cases} t & n < n_0 \\ aT(\frac{n}{b}) + f(n) & \text{otherwise} \end{cases}$$

the variable t relates to the running time of the direct phase.
$\frac{n}{b}$ relates to the size of the problem after it is divided,
the multiple of a allows for performing the algorithm on a instances of the subproblems.
$f(n)$ allows for the running time of the combine step.

(e) [4 marks] In what situation might the Master Method not apply to the recurrence of **(d)**.
Answer:
The Master Method might not apply to the recurrence of **(d)** in two situations,

1) if $f(n)$ is smaller than $n^{\log_b(a)}$ but not polynomialy so.

2) if $f(n)$ is larger than $n^{\log_b(a)}$ but not polynomialy so.

continued...

Question 4. [50 marks]

Consider a manager of a timber company. He is managing the destruction of a forest containing trees of different types. Different types of wood come in different size logs and sell for different values, for example: Rimu comes in 150 kg. logs and sells at \$1500 per log, Teak comes in 60 kg. logs and sells at \$550 per log. The table below gives a list of the weights and values of four types of wood in the forest.

Wood type	weight	value	value/kg
Rimu	150 kg.	\$1500	\$10
Teak	60 kg.	\$550	\$9.17
NZ. Pine	100 kg.	\$500	\$5
Macro-Carpa	60 kg.	\$400	\$6.67

The wood must be transported from the forest to the towns where it is to be sold. Each truck may take a total load which is a maximum of 50,000 kg. The manager aims to maximise the value of the timber on each load.

He wrongly assume that he can use the following greedy algorithm to achieve his optimal sale.

> **Greedy:** Look at each log in decreasing order of value per kg., selecting the log only if it will fit on the truck load.

(a) [4 marks] Define **acceptable solution** and **correct solution** for the general problem: that is, assuming that the values n and W and the arrays w and v are inputs, where n is the number of types of wood, W is the maximum load allowed in the truck, w records the weights of the different types of wood on the truck and v records the value of the different types of wood on the truck.

Answer:
An acceptable solution occurs when the sum of the weights held on the truck $< W$.
A correct solution occurs when the sum of the values relative to v is the greatest.

(b) [4 marks] Show that the greedy algorithm does **not** give the optimal solution by comparing the truck load given by the managers algorithm with a better one (with a higher value). Assume that the truck can carry the maximum weight limit (50,000 kg.).

Answer:
Using the greedy algorithm we get a load of 333 logs of Rimu, with weight $333 * 150\text{kg.} = 49,950\text{kg.}$ and value \$499,500.
An optimal load would be 332 logs of rimu plus 3 logs of teak, weighing $332 * 150\text{kg.} + 3 * 60\text{kg.} = 49980\text{kg.}$ and with value 332*\$1500+3*\$550=\$499650

(c) [8 marks] Show that the general problem has the **optimal substructure** property.

Answer:
We define the general problem (w, v, n, W) with the above meanings. If we have an

continued...

optimal solution to the problem $Sol = (w, v, n, W)$ and w has i elements (i logs). Then define a sub-problem $sol' = (w, v, n-1, W')$ where $W' = W - w_i$. (i.e. take off one log.) It is clear that sol' must be an optimal solution to the problem with $n-1$ logs and W' weight limit since if it wasn't, then this would mean there was another solution, which adding log w_i to would give a *more optimal* solution than *sol* - contradiction since *sol* is defined as optimal.

(d) [8 marks] Describe an appropriate **dynamic programming** algorithm for solving the general problem.

Answer:

```
$truck \leftarrow$ new array(4) // holds the numbers of each type of log
$K_v \leftarrow$ new array(W,4) initialised to 0
$K_w \leftarrow$ new array(W,4) initialised to 0
for $i$ in 1 to W repeat
  for $j$ in 1 to 4 repeat
    if $K_w(i-1,j) + w(j) < i$ then
      $k_w(i,j) \leftarrow k_w(i-1,j) + w(j)$
      $k_v(i,j) \leftarrow k_v(i-1,j) + v(j)$
      $truck(j) \leftarrow truck(j) + 1$
    else if $K_v(i,j-1) > K_v(i-1,j)$ then
      $k_w(i,j) \leftarrow K_w(i,j-1)$
      $k_v(i,j) \leftarrow K_v(i,j-1)$
      $truck(j) \leftarrow truck(j-1)$
return $truck$
```

(e) [6 marks] Briefly outline a proof that your algorithm is correct.

Answer:
Determine the *optimal substructure* in the problem.
Prove that the implementation satisfies the optimal substructure property for every subproblem in the subproblem sequence.

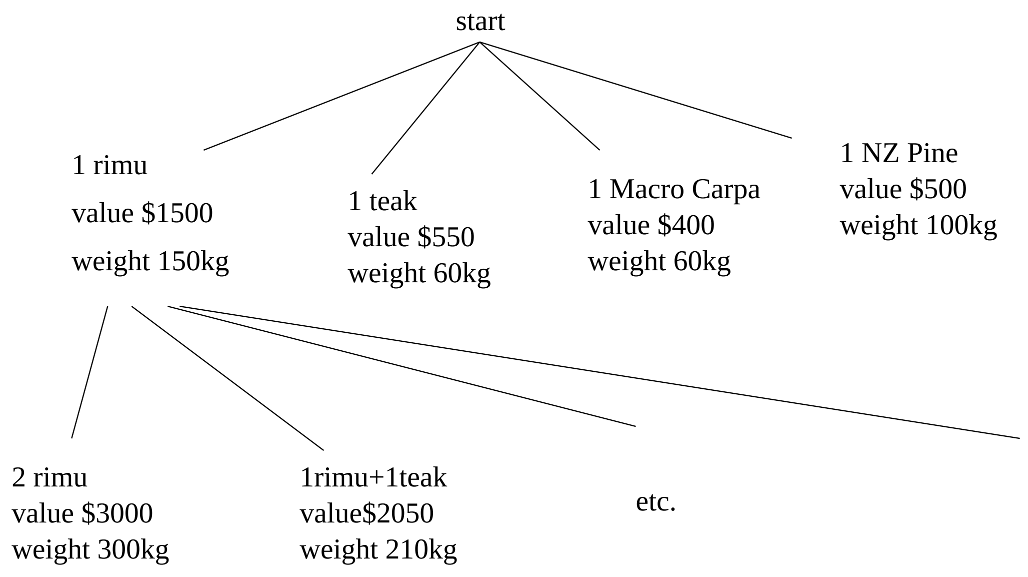
(f) [4 marks] State the asymptotic complexity of your algorithm. **Justify your answer.**

Answer:
$\Theta(Wn)$ where $n = 4$ in this case

Another approach to the managers problem is to use a **backtracking search**.

(g) [8 marks] Describe a backtracking search algorithm for the general problem. You may find it useful to illustrate your algorithm using the example problem instance above.

Answer:
It is possible to build a tree, such that every node corresponds to different configurations of the trucks load for the above problem. Each node will have the numbers of each log, the total value and the total weight. The tree would look a bit like:



This is an implicit tree which arrises from a recursive depth first search. The termination conditions (determining the leaf nodes of the tree) is that the weight attributes of the nodes $\geq$ the weight limit of the truck. The backtracking search algorithm will do a depth first search of the tree, for the maximum value attribute.

(h) [8 marks] Describe a related problem that gives an upper bound on the value of a solution to the problem, and explain how solutions to the related problem may be used to improve the efficiency of the backtracking search.
Answer:
A related problem which gives an upper bound on the value of a solution to the problem is what is known as the $(0,0)$-knapsac problem. In this particular instance it would allow us to take fractions of a logto fill up our truck load. So in this case, the truck would only take Rimu loads, as it is the most valuable. How this helps with the general problem is that it gives us an upper bound to the solution, which allows us to do quite considerable *pruning* of the search tree, which means that we don't need to consider so many nodes.

Question 5. [23 marks]

(a) [5 marks]
Describe, in informal terms, the complexity classes $\mathcal{P}$, $\mathcal{NP}$ and $\mathcal{NP}$-**Complete**.

Answer:
The complexity class $\mathcal{P}$ denotes the set of problems which may be solved by a *polynomial time* algorithm. That is the set of problems solvable by Turing computable algorithms whose completion time can be bounded by a polynomial.

The complexity class $\mathcal{NP}$ stands for *Non-deterministic Polynomial* and denotes the set of problems which may be solved by algorithms computable on a non-deterministic Turing machine whose completion time may be bounded by a polynomial.

The complexity class $\mathcal{NP}$-**Complete** contains the hardest of all $\mathcal{NP}$ problems.

(b) [8 marks] Draw a Venn diagram showing the following complexity classes, you should use an arrow to indicate the increase in asymptotic complexity: (Assume that $\mathcal{P} \neq \mathcal{NP}$)

(i) $\Theta(n\log(n))$

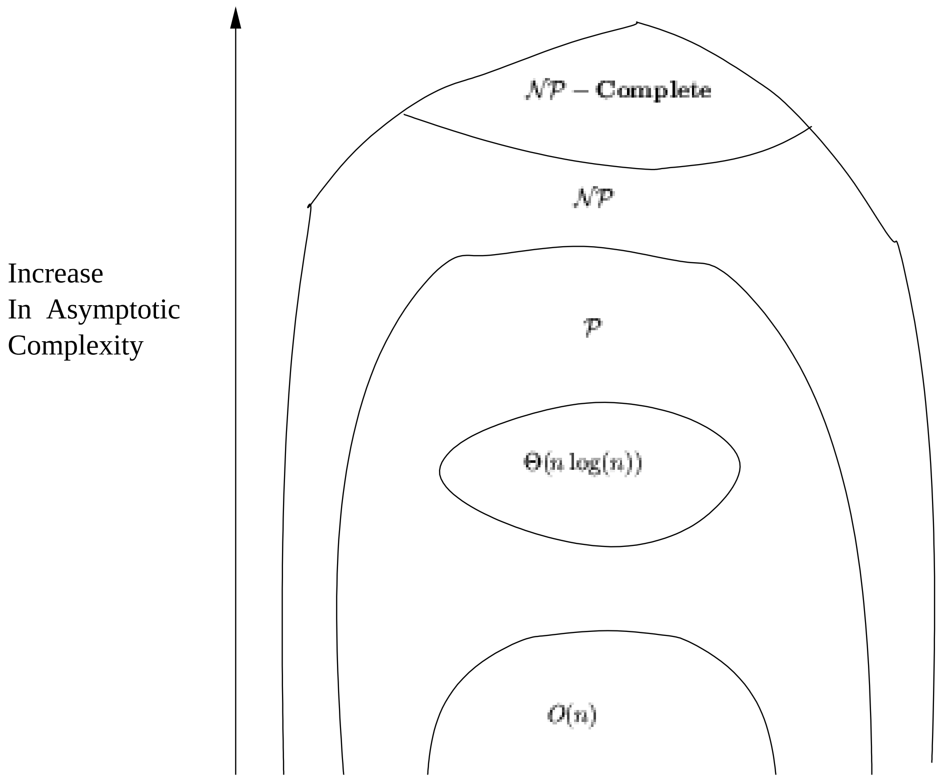
(ii) $O(n)$

(iii) $\mathcal{P}$

(iv) $\mathcal{NP}$

(v) $\mathcal{NP}$-**Complete**.

Answer:



continued...

(c) [6 marks] It is an accepted fact that the set of algorithms is not large enough to cover the set of problems. A way of proving this fact is to produce a problem for which there can not be an algorithm.
Describe a problem which is not computable. Show why it is not computable.
Hint: Base the problem Turing's famous *halting problem*.

Answer:
First define a problem *halt*

```
halt
input: f, The encoding of an algorithm
output: true or false depending on whether the algorithm terminates
If f terminates
   return true
else
   return false
```

Now define a second problem H'

```
H'
input: f, The encoding of an algorithm
output: void
if halt(f) then
   loop forever
else
   terminate immediately
```

Now run H' on itself, note that if it does terminate, *halt* returns true, so it loops forever, otherwise *halt* returns false, so it terminates immediatley. So either way we loose!

(d) [4 marks] Suppose that q is an $\mathcal{NP}$-**Complete** problem and we have found a polynomial time algorithm for solving q. What important result may we infer regarding the algorithmic categories mentioned in question **(a)**. Justify your answer.
Answer:
Since we have found a polynomial time algorithm for an $\mathcal{NPC}$ problem q, $q \in \mathcal{P}$. Then one of the hardest $\mathcal{NP}$ problems is in $\mathcal{P}$. That means that all $\mathcal{NP}$ problems are, so $\mathcal{NP} \subseteq \mathcal{P}$. Therefore since $\mathcal{P} \subseteq \mathcal{NP}$, $\mathcal{P} = \mathcal{NP}$.

continued...

Question 6. [28 marks]

(a) [6 marks] Using pseudocode or otherwise, describe a generic Las Vegas algorithm.
Answer:

```
Las Vegas
repeat
  answer ← solve problem with some probability (small for efficiency)
  if answer is correct
    return answer
  else
    iterate
```

(b) We wish to check whether two large structures have equal values. We may do this using an algorithm, *probeq* which is based on random sampling. If the answer given by *probeq* is *false*, it must be correct. If the answer is *true*, there is a chance of $\frac{1}{4}$ (one out of four) that it is incorrect. We would like to reduce the possibility that we get an incorrect answer. We discover that we may do this by repeatedly calling *probeq* using the following algorithm, *ProbEqPlus*:
(we may assume that different calls to *probeq* are independent.)

```
ProbEqPlus
Input: Two values $x$, $y$ and $n$, a loop bound
Output: $true$ or $false$ depending on whether $x = y$
for $i$ in 1 to $n$ repeat
  if not probeq(x,y) then
    return false
return true
```

We must supply the loop bound n as a parameter to *ProbEqPlus*. This will determine the probability with which the answer returned by the algorithm is correct.

(i) [1 mark] Suppose the value returned by *ProbEqPlus* is *false*. What is the probability that this is correct?

Answer:
1 that is 100%

(ii) [3 marks] Suppose we call *ProbEqPlus* with $n = 5$ and the value returned is *true*. What is the probability that this is incorrect?

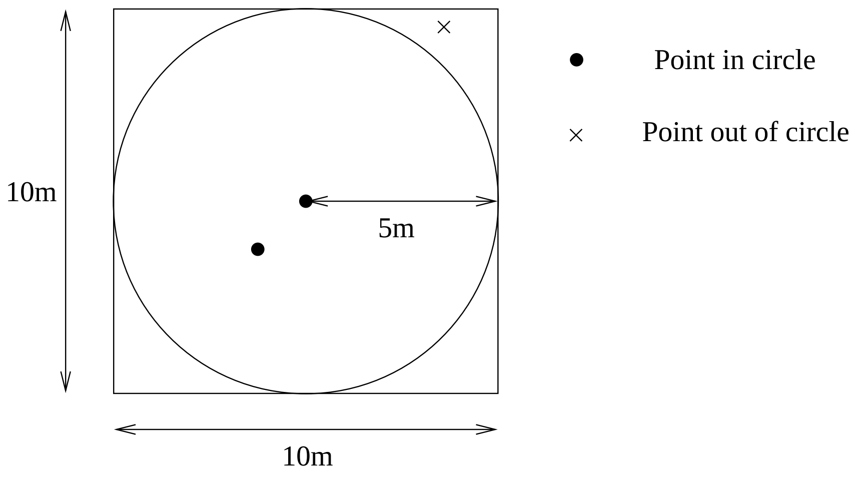
Answer:
$\frac{1}{4^5} = \frac{1}{1024}$

(iii) [4 marks] What value of n would ensure that if *ProbEqPlus* returns *true*, then the probability that this is incorrect is less than 2^{-64}? Justify your answer.

Answer:
$n = 33$ since $\frac{1}{4^{33}} < 2^{-64}$. N.B. $\frac{1}{4^{32}} = 2^{-64}$, so this is not good enough!

(iv) [4 marks] Name another example of a Monte Carlo algorithm that you have studied. What are the subproblems in this case?
Answer:
probably give Miller/Rabin primality testing, with pseudo primality

(c) One day you are on the beach with some friends. You decide to do a bit of mathematical investigation. You draw a circle and a square in the sand as shown below. You then collect ten stones and start throwing them (randomly) at the square. Some land in the circle, some land outside the square and some land inside the square but outside the circle. If a stone lands outside the square it must be thrown at the square again until it lands inside it.
You find that 8 out of 10 stones land inside the circle.



(i) [3 marks] By considering the ratio of how many stones land inside the square in total and how many land inside the circle, explain how to calculate an approximation to the mathematical constant π, justifying your answer.

Answer:
Let

T = total area of square

C = area of circle

$$\frac{T}{C} = \frac{(2r)^2}{\pi r^2} = \frac{4r^2}{\pi r^2} = \frac{4}{\pi} \approx \frac{T_n}{C_n}$$

where

T_n is the number of stones landing in the square

C_n is the number of stones landing in the circle

therefore $\pi \approx \frac{4C_n}{T_n}$

(ii) [2 marks] By considering your results calculate an approximation to π.

Answer:

$$\pi \approx \frac{4 * 8}{10} = 3.2$$

continued...

(iii) [5 marks] Using pseudocode, describe a probabilistic numeric algorithm that may be run on a computer to approximate π using this method.

Answer:

```
*approxPi:*
**intput:** $b$ the number of stones we have.
**output:** our appoximation to $\pi$
count$\leftarrow$ 0
for i$\leftarrow$ 1 to b repeat
   $x\leftarrow$ random(0..1)
   $y\leftarrow$ random(0..1)
   if $(x-0.5)^2+(y-0.5)^2<0.25$ – stone lands inside circle
      count$\leftarrow$ count+1
return 4*count/b
```

Hint:
1) the area of a circle $=\pi r^2$ where r is the radius of the circle.
2) Pythagoras theorem says that for a right angled triangle $r^2=x^2+y^2$ where x and y are the sides at right angles to each other and r is the hypotenuse (the longest side).

****