search algorithm

research	5		
class design	4		
bow line generation	2		
~~box~~ area check & length check	1		
~~length check~~	~~1~~ 2		
bow point generation	1		
min/max width/height determination for bow point	2		
bow line symmetry check	2		
search sequencing	3		
initial investigation (already done)	~~2~~ 2		
	23	23	+100% (46 M)

database

initial investigation (already done)	3		
download JDBC driver & try using it	4		
download trial version of Paradox & try it	2		
JDBC familiarization	2		
boat lookup	2		
	13	13	+20% (16 M)

logging

	2	2	+20% (2 M)

GUI

boat entry dialog	5		
queue display / editing	4		
graphical display of lock / boats	10		
log request / display	4		
menus & general organization	5		
	28	28	+50% (42 M)

other

system testing	10		
visits to Marina	4		
customer spec changes / additions	10		
user documentation	4		
proposal & quote (done)	2		
license agreement (done)	3		
	33	33	+20% (40 M)
		99	146 M

= 20 weeks
= 5 months.
(= 29 weeks
= 7.2 months
M)

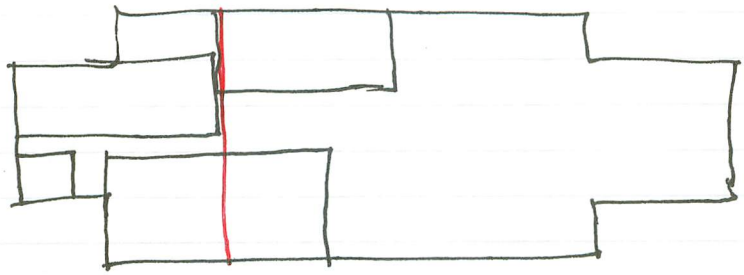
100 days @ £300 = £30000 (£40/hour)
100 days @ £200 = £20000 (£27/hour)
100 days @ £150 = £15000 (£20/hour)

lock packing

~~it should be possible to~~

P1 • proposition: the lock can be filled in "bow order". i.e. ~~always each~~ as each boat is positioned in the lock, its bow will not be further forward than ~~any boat~~ the bow of any boat that is already in the lock. [see CE1.1]

if this proposition is correct, then any space ahead of the bow of the last boat entered is "dead space" — i.e. we do not need to consider putting boats into it.

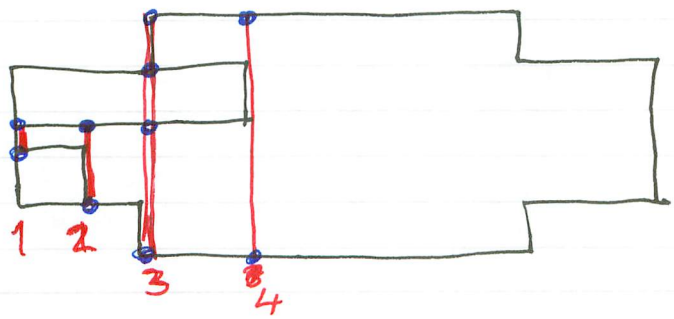


e.g. space to the left of the red line is dead (assuming we are filling from left to right).

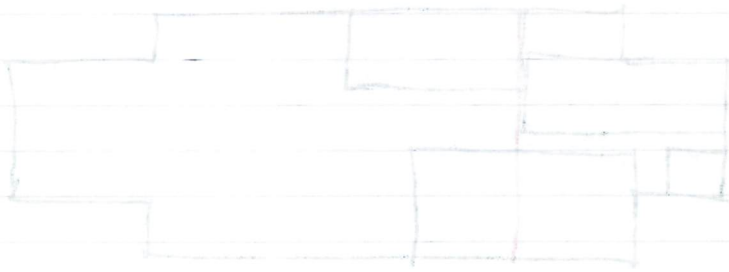
P2 • proposition: the bow of each boat will be positioned against:
(a) the exit lock gate.
or (b) a step in the lock side,
or (c) the ~~bow~~ stern of another boat.

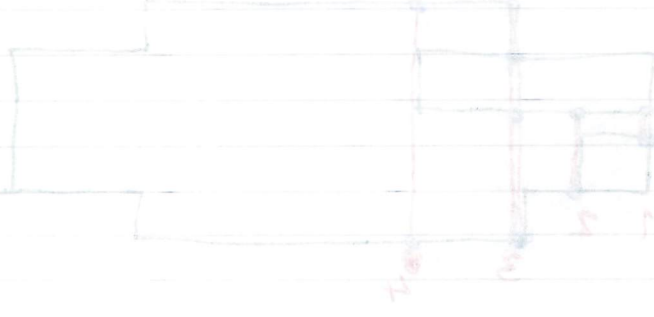
P3 • proposition: one side of each boat will be positioned against:
(a) the lock side.
or (b) a side of another boat.

P4 • proposition: we can consider possible bow positions in left-to-right order.
e.g.:



we try all possible boats at lie 1. if none fit, we move to lie 2 (and ~~consider~~ space to the left of lie 2 becomes dead), and so on.
note that the red lines are determined by P2. call these red lines "bow lies".



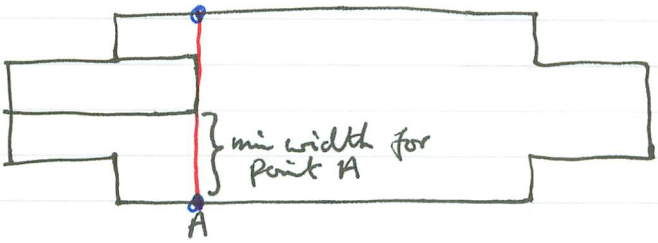


Lock packing

P5. proposition: each bow line has points on it where the left or right side of a bow can be positioned, and we need consider only these positions.
e.g. marked in blue on diagram above.
these points are determined by P3. i.e. we need to find intersections of the bow line with boat sides and lock sides, but only include intersections with boat sides where the boat side extends to the right of the bow line. call these blue points "bow points". [see EP5.1]

P6. proposition: each bow point has a maximum width of boat that can be positioned at it.

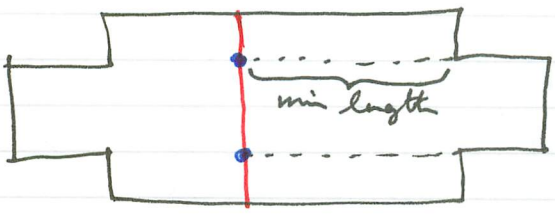
P7. proposition: a bow point may have a minimum width of boat that can be positioned at it.
e.g.:



the minimum width is a consequence of P2.

P8. proposition: a bow point has a maximum length of boat that can be positioned at it.
this is determined by the entrance lock gate or a step in the lock side, whichever is in line with the point.

EP5.1 the steps & the lock sides near the entrance lock gate introduce two additional bow points that need to be considered for large boats:

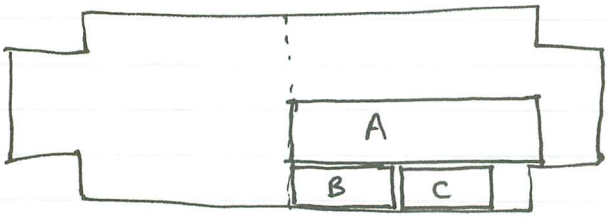


P9. proposition: a bow point may have a minimum length of boat that can be positioned at it.
e.g. see diagram above.

lock packing

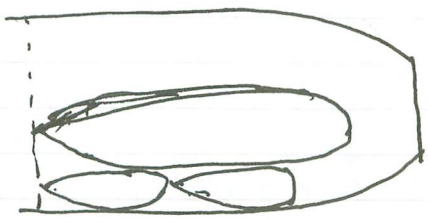
P10. proposition: if a boat satisfies the min and max width and length requirements for a bow point, then it will fit at that bow point.
i.e. no further 'fit' tests need to be done.

CE1.1. a boat may restrict access if its stern is in the narrow part of the lock near the entrance lock gate:



the "bow order" of filling would put the boats in the order ABC or BAC, but ~~[illegible]~~ boat C (and boat B in the ABC order) would not be able to get in past boat A. ~~three~~ two possible ways of dealing with this problem:

1. abandon "bow order" as a way of ordering the search, and try to find some other way of organizing the search — I'm very reluctant to do this, since the left-to-right progression of bow lines seems to have significant advantages.
2. use "bow order" for ordering the search, but once an organization has been found use a separate algorithm to find a fill order.
3. assume that in practice boats B and C would be able to get past boat A, because the lock and the boats are curved, not rectangles:



4. don't attempt to generate a fill order — the spec doesn't actually ask for it — and let the user determine it.

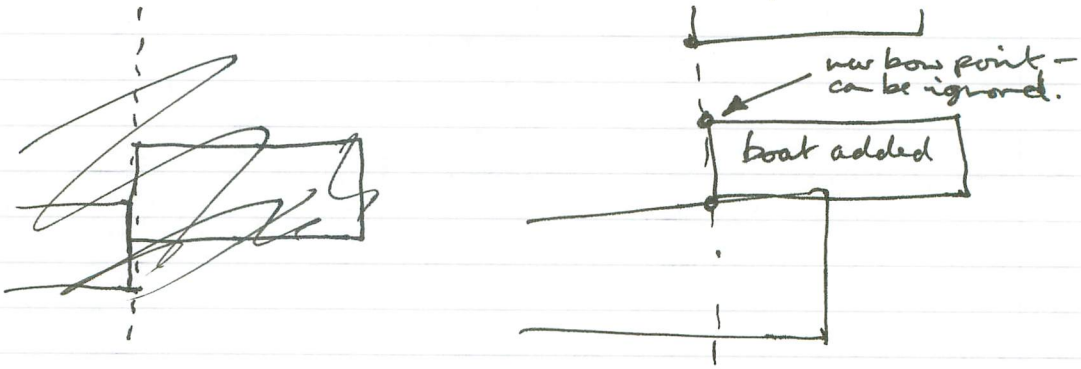
P11. proposition: as each boat is added it will generate at most one new bow line.
the ~~two~~ new bow line will be at the stern of the boat, and will new if it does not coincide with the stern of another boat.

P12. proposition: as each boat is added it will generate at most one new bow point on the current bow line.
the bow point will be ~~[illegible]~~ at the opposite side of the bow from the the bow point to which the boat was added.

A
B
C

lock packing

P13. proposition: the bow points on a bow line can be tried in top to bottom order.
i.e. when we add a boat to a bow point we do not need to reconsider any unused bow points above that bow point. also, if adding a boat to a bow point generates a new bow point above that bow point then we do not need to consider that new bow point:



P14. proposition: the bow points on a bow line occur in pairs, corresponding to positioning a boat at the left (bottom) or right (top) of a space.

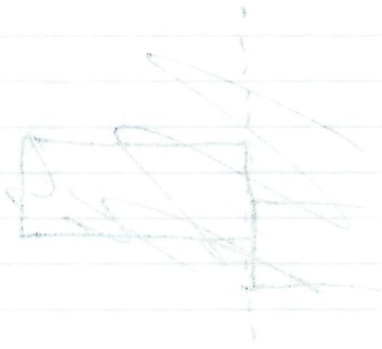
P15. proposition: we can reject a partial solution if the area left is less than the total area of the remaining boats.
the area left is the area to the right of the current bow line less the area of any boats or parts of boats to the right of the bow line.

P16. proposition: we can reject a partial solution if the distance from the current bow line to the entrance lock gate is less than the length of a remaining boat.

P17. proposition: the area check suggested by proposition P15 only needs to be applied when we move to a new bow line, adding a boat does not affect the check, since the area left and the total area of the remaining boats is reduced by the same amount.

P18. proposition: considering boats in decreasing order of their area will reduce the search space, since the area check will fail earlier. (note that the length check of P16 copes fairly well with the most obvious examples where this heuristic might help).

P19. proposition: if a bow line and its set of bow points is symmetrical, then we need consider only half the bow points.
actually, the test of symmetry will also need to take into account parts of boats that extend to the right of the bow line.
this rule can only be applied when we first move to a new bow line.



lock packing.

size of search space:

N	N!	3^N	$N! \cdot 3^N$
2	2	9	18
3	6	27	162
4	24	81	2000
5	120	243	30000
6	720	729	500,000
7	5040	2241	10,000,000
8	40320	6723	300,000,000
9	362,880	20,169	7,000,000,000
10	3,628,800	60,507	200,000,000,000
11	36,916,800	181,521	6,000,000,000,000
12	413,001,600	544,563	200,000,000,000,000

at each bow point, we have a choice of using that bow point or not, and, if we choose to use it, we have a choice of which boat to put there. many of the bow points will not allow any boat to be positioned there, and so will not add to the search space. a bow point will only add to the search space if it is possible to put a boat there and we also have to consider not putting a boat there.
in most cases, all the boats that can be positioned at the top bow point of a pair can also be positioned at the bottom point of the pair. hopefully, whenever we have tried all (or most) of the boats at a pair of bow points, the search space arising from not putting a boat at the bow point pair will be severely pruned on grounds of area. so, optimistically, we would only have 2 bow points to consider at each stage in the search — but let's say 3 bow points, to be more realistic.

a 1GHz CPU can perform roughly 1,000,000,000 instructions per second. a step of the search is likely to need at least 1000 instructions — let's say 10,000 to be on the safe side. so we can perform roughly 100,000 search steps per second. this suggests that N=7 (100 seconds) would the limit for exhaustive searching.