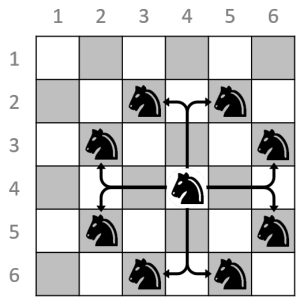
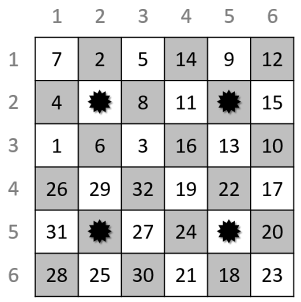
Homework #1

(Due: Oct 6)



(*a*) All possible moves of a knight on a 6×6 chessboard from location $(4, 4)$



(*b*) A closed knight's tour on a 6×6 chessboard with holes

Figure 1: Knight's tour on a chessboard with holes.

Task 1. [**50 Points**] **Knight's Tour on a Defective Chessboard**

Suppose you are given an $n \times n$ chessboard for some $n \geq 3$. Each cell is identified with a pair (r, c), where $r \in [1, n]$ is its row number and $c \in [1, n]$ is its column number. However, some of the cells of the chessboard are defective (i.e., have holes in them), and so are unusable.

A *knight* is a chess piece that moves in an 'L' shape in any direction (see Figure 1(a)). A *knight's tour* on this chessboard is a sequence of moves made by the knight starting from a good (i.e., nondefective) cell and touching every other good cell exactly once. If the last cell on the tour is reachable from the first cell in a single move of the knight, we call the tour *closed*. Figure 1(b) shows a closed tour on a 6×6 chessboard with 4 defective cells, where each good cell is marked with an integer giving its rank[1] in the tour. The tour starts at cell $(3, 1)$ and ends at cell $(4, 3)$, and it is closed because cell $(3, 1)$ can be reached from cell $(4, 3)$ in a single knight's move.

(a) [**25 Points**] Design and explain a recursive divide-and-conquer algorithm for finding a closed knight's tour on a defective $n \times n$ chessboard with $n = 3 \times 2^k$ for some integer $k \geq 0$. Assume that for $0 \leq i, j < 2^k$, cells $(3i + 2, 3j + 2)$ have holes. Include pseudocode.

(b) [**10 Points**] Write down a recurrence describing the running time of your algorithm from part (a), and solve it.

(c) [**15 Points**] Repeat parts (a) and (b) for an $n \times n$ chessboard with $n = 3 \times 2^k$ for some integer $k > 0$. Assume that this time the chessboard has no defective cells.

[1]'rank' of a cell is equal to the number of cells visited before that cell plus one

Task 2. [50 Points] Distance-Incorporated Codon Autocorrelation Score

The availability of synonymous codons (codons that can translate the same amino acid into a protein) enables a protein to be encoded by many different sequences of codons/tRNAs. Autocorrelation measures the reuse of a particular codon/tRNA in succession (instead of choosing a different synonymous one) during the translation of a protein sequence. Studies show that tRNA autocorrelation in a coding sequence has important effects on its translation speed.

We can measure autocorrelation by transforming the problem into a combinatorial one. For example, suppose we have a sequence of amino acids where Serene's residues occur at positions 4, 6, and 301, and to be coded by a mix of codons from two different tRNAs (say, two of type A and one of type B). These codons can appear in three different relative orders: AAB, BAA, and ABA. Considering the opportunities of tRNA reuse, AAB is expected to translate faster and is more autocorrelated than BAA, because the two occurrences of A in AAB are so close that the specific tRNA molecule employed in coding for the 4th residue is likely to be around when residue at position 6 is being translated, and hence can be reused.

Distance Incorporated Codon Autocorrelation (DICA) score is a recently proposed metric of gene autocorrelation which is calculated for a coding sequence by finding positions of all synonymous codons for a given amino acid and then summing a reward function, $F(d(i,j))$ which assigns a positive score based on the distance between the synonymous codons. Here, $d(i,j)$ is the distance between codons translated by the same tRNA, i.e., if a tRNA repeat is found at positions i and j then the distance between these two is $d(i,j) = j - i$. Because autocorrelation appears to decay slowly with distance, for a given synonymous codon, the probability that the next codon is the same decreases as the distance increases. It has been shown that an exponential distance function of the form $c^{d(i,j)}$ where $c \approx 0.999$ gives a better measure for DICA.

Suppose we have a codon sequence $\mathcal{S}$ of length n composed of K (≤ 9) different amino acids[2] $\mathcal{A}_a$, $1 \leq a \leq K$. Let $\mathcal{S}_a$ be the maximal subsequence of $\mathcal{S}$ containing only $\mathcal{A}_a$'s, and let n_a be its length. Then $n = \sum_{a=1}^{K} n_a$, and clearly, $n = \Theta\left(\max_{a=1}^{K} n_a\right)$. For any given $a \in [1, K]$ and $1 \leq i \leq n_a$, let $l_i^{(a)}$ be the location of $\mathcal{S}$ where the i-th entry of $\mathcal{S}_a$ occurs. Then the *Distance-Incorporated Codon Autocorrelation* (DICA) score ($D_{\mathcal{S}}$) for $\mathcal{S}$ is:

$$D_{\mathcal{S}} = \frac{\sum_{a=1}^{K} \sum_{i=1}^{n_a} \sum_{j=i+1}^{n_a} \theta(T_{l_i^{(a)}}, T_{l_j^{(a)}}) \times F(d(l_i^{(a)}, l_j^{(a)}))}{\sum_{a=1}^{K} \sum_{i=1}^{n_a} \sum_{j=i+1}^{n_a} F(d(l_i^{(a)}, l_j^{(a)}))},$$

where T_i is the tRNA at location i of $\mathcal{S}$, $\theta(T_i, T_j) = 1$ if $T_i = T_j$ and 0 otherwise. Note that there can be at most m different tRNA's, where $m \leq 6$.

This equation can be evaluated using three nested loops. However, such a naïve computation of DICA takes $\Theta\left(n^2\right)$ time, which can be prohibitively expensive for very long sequence.

Show how to compute $D_{\mathcal{S}}$ for a sequence of length n in $\Theta\left(n \log n\right)$ time. Include pseudocode of your algorithm.

[2]There are 9 amino acids with synonymous codons translated by different tRNAs (*A, G, I, L, P, R, S, T, V*)

```
SELECT( $A[q:r],\ k,\ d,\ s_{even},\ s_{odd},\ b$ )
Input: An array of distinct elements, and an integer $k \in [1, r-q+1]$. The parameter $d$ is the
depth of recursion with $s_{even}$ being the block size to be used at even depths and $s_{odd}$ at odd depths.
Also $b$ is an upper bound on the size of the base case.
Output: An element $x$ of $A[q:r]$ such that $rank(x, A[q,r]) = k$.
 1. $n \leftarrow r - q + 1$
 2. if $n \leq b$ then
 3.     sort $A[q:r]$
 4.     return $A[q+k-1]$
 5. else
 6.     if $d \bmod 2 = 0$ then $s \leftarrow s_{even}$
 7.     else $s \leftarrow s_{odd}$
 8.     divide $A[q:r]$ into blocks $B_i$'s each containing $s$ consecutive elements
        ( last block may contain fewer than $s$ elements )
 9.     for $i \leftarrow 1$ to $\lceil \frac{n}{s} \rceil$ do
10.         $M[i] \leftarrow$ median of $B_i$ using sorting
11.     $x \leftarrow$ SELECT$\left( M\left[1 : \lceil \frac{n}{s} \rceil\right],\ \left\lfloor \frac{\lceil \frac{n}{s} \rceil + 1}{2} \right\rfloor,\ d+1,\ s_{even},\ s_{odd},\ b \right)$   {median of medians}
12.     $t \leftarrow$ PARTITION( $A[q:r],\ x$ )        {partition around $x$ which ends up at $A[t]$}
13.     if $k = t - q + 1$ then return $A[t]$
14.     else if $k < t - q + 1$ then return SELECT( $A[q:t-1],\ k,\ d+1,\ s_{even},\ s_{odd},\ b$ )
15.         else return SELECT( $A[t+1:r],\ k,\ d+1,\ s_{even},\ s_{odd},\ b$ )
```

Figure 2: Selection with hybrid blocking.

Task 3. [60 Points] Recursive Selection with Hybrid Blocking

Figure 2 shows a slightly generalized version of the selection algorithm we saw in the class. Instead of using a single block size (e.g., 5) at all levels of recursion, it uses block size s_{even} at even levels, and s_{odd} at odd levels. Now the base case size b is also a parameter to the algorithm. Observe that when $b = 140$ and $s_{even} = s_{odd} = 5$, the algorithm reduces to the one we saw in the class.

(*a*) [**10 Points**] Write a recurrence relation describing the running time of SELECT on an array of size n assuming $s_{even} = s_{odd} = 3$. What is the best running time you get by solving the recurrence? What is the smallest value of b you get?

(*b*) [**20 Points**] Repeat part (*a*) with $s_{even} = 3$ and $s_{odd} = 5$.

(*c*) [**30 Points**] Suppose we run SELECT with $s_{even} = s_{odd} = 4$. Then in steps 8–10, each group of size 4 will have exactly 2 medians — one smaller and one larger. Suppose in step 10, we assign the smaller median to $M[i]$. Then what will be the running time of SELECT on an array of size n? If the running time is $\omega(n)$, can one make change(s) to steps 11–15 to bring the running time down to $\Theta(n)$?