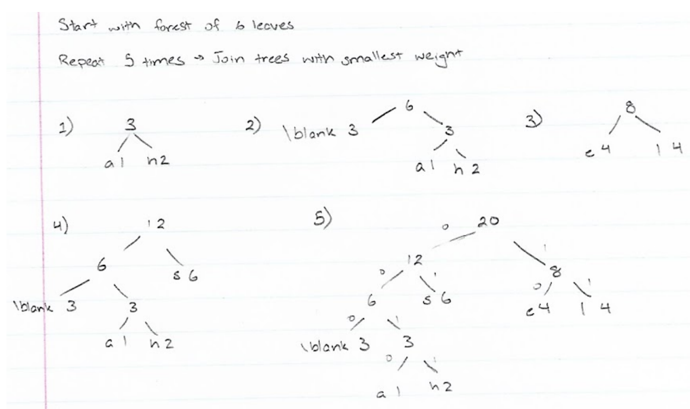
Dennis Kuzminer
CSCI-UA 310-001 PS5

1.
    a.

Character	Frequency
s	6
h	2
e	4
\blank	3
l	4
a	1
Total	20

    b.



    c.

Character	Code
s	01
h	0011
e	10
\blank	000
l	11
a	0010

she sells sea shells →
01001110000011011110100001100010000010011101111101

Dennis Kuzminer
CSCI-UA 310-001 PS5

2.

a. n > m → The number of jobs is more than the number of machines.
If there are m machines, and m is strictly less than n, then there must be at least one machine with 2 jobs (pigeonhole principle). If we order in descending order, then the first job to be assigned to a machine with a pre-existing job is the m + 1 job. Since the ordering is descending, we know that the time of the m + 1 job (t_{m+1}) is at least as great as all of the jobs that came before it $(t_1 \geq t_2 \geq ... \geq t_{m+1})$. Because there are now at least 2 jobs assigned to this machine, and the first job was at least as long as the second, we can say that $T^* \geq 2t_{m+1}$.

b. Let *l* be the last job scheduled on machine *k*. Right before l was scheduled: the load on machine k was $T_k - t_l$ and the load on k was minimal.

From class, we derived that $T_k \leq t_l + \frac{1}{m}\sum_j t_j$. We also know that $2t_{m+1} \leq T^* \rightarrow$
$t_{m+1} \leq T^*/2$. Since l was the last job, we can say that $t_l \leq t_{m+1}$.
Combining: $t_l \leq t_{m+1} \leq T^*/2$, showing that t_l is bounded ($t_l \leq T^*/2$).

By Lemma 2, $\frac{1}{m}\sum_j t_j$ is also bounded $\frac{1}{m}\sum_j t_j \leq T^*$.
Substituting: $T_k \leq T^*/2 + T^* \rightarrow T \leq 3T^*/2$

Dennis Kuzminer
CSCI-UA 310-001 PS5

3.

a. $Opt(i)$:
if $i = 0$ then
    $result \leftarrow 0$
else
    $result \leftarrow \infty$
    for k in $[0 .. i)$ do
        $penaltyForLastDay \leftarrow (200 - (a_i - a_k))^2$
        $penaltyForPreviousDays \leftarrow Opt(k)$
        $result \leftarrow \min(result, penaltyForLastDay + penaltyForPreviousDays)$
return *result*

b. $Opt(i)$:
if $T[i] = \perp$ then
    if $i = 0$ then
        $T[i] \leftarrow 0$
    else
        $T[i] \leftarrow \infty$
        for k in $[0 .. i)$ do
            $penaltyForLastDay \leftarrow (200 - (a_i - a_k))^2$
            $penaltyForPreviousDays \leftarrow Opt(k)$
            $T[i] \leftarrow \min(result, penaltyForLastDay +$
                $penaltyForPreviousDays)$
return $T[i]$
This algorithm will execute operations up to n for each n, making run time without memoization n(n+1)/2 or just O(n^2).

c. $Opt(n)$:
for i in $[0 .. n)$
    $T[i] \leftarrow 0$ // default value
for i in $[0 .. n)$
    $result \leftarrow \infty$
    for k in $[0..i)$ do
        $penaltyForLastDay \leftarrow (200 - (a_i - a_k))^2$
        $penaltyForPreviousDays \leftarrow T[i]$
        if $penaltyForLastDay + penaltyForPreviousDays < result$
            $result = penaltyForLastDay + penaltyForPreviousDays$
    $T[i] = result$
return $T[i]$
For the same reason as in (b), we can see that this is also O(n^2)

d. $Sol(i)$:
if $i = 0$ then
    return emptyList()

else

for k in [0 .. i) do

if $T[i - k] + (200 - (a_i - a_k))^2 = T[i]$ then

return $concat(Sol(i - k), k)$

This algorithm can be implemented in linear time (O(n)), if the concat operation is also linear. This can be achieved with a linked list data structure storing out answer.

Dennis Kuzminer
CSCI-UA 310-001 PS5

4. *Opt*(t):
   if $T[t] = \perp$
       if $|t| = 0$
           $T[t] \leftarrow$ true
       else
           for i in $[0..k)$
               if t.substring$(0, |s_i|) = s_i$ and $Opt(t.\text{substring}(|s_i|, \text{t}))$
                   $T[t] \leftarrow$ true
               else
                   $T[t] \leftarrow$ false
   return $T[t]$
   Invoke Opt(t) and have the set of s tiles as a global variable

   The running time would be $O(\,|t| * \sum_j |s_j|\,)$, as for the length of t, we consider subproblems where we compare all tiles. This would optimized for runtime if we can implement substring in a contant time.

Dennis Kuzminer
CSCI-UA 310-001 PS5

5. First, we create an m by n matrix where we create an extra row and column for gaps. Fill each of the entries in that column and row with the score of the gap (0, 1, 2, 3 ....) and continue to increment down for all n and m.

Dennis Kuzminer
CSCI-UA 310-001 PS5

6. $Opt(t_0 \ldots t_j)$:
   $T[0] = t_0$
   for j in $[1..n]$
   $\quad$ $T[j] \leftarrow \max(t_j)$

Solution to 3c that is actually greedy and doesnt work as well

```
        Opt(n) :
incr = 0
optPenalty = 0
for i in [0 .. n) incrementing i by incr do
        minPenalty = ∞
        for k in [0..n) do
                if 200 – (a_k – a_i))^2 < minPenalty
                        incr = k - i
                minPenalty = min( minPenalty , (200 – (a_k – a_i))^2 )
        optPenalty += minPenalty
return optPenalty
```