Name:

SID:

Name and SID of student to your left:

Name and SID of student to your right:

Exam Room:

○ Evans 60 ○ Kroeber 160 ○ Latimer 120 ○ North Gate 105
○ Pimentel 1 ○ Cory 293 ○ Soda 320 ○ Soda 380
○ Other

Assigned Seat:

Rules and Guidelines

- **The exam has 16 pages, is out of 110 points, and will last 110 minutes.**
- Answer all questions. Read them carefully first. Not all parts of a problem are weighted equally.
- Write your student ID number in the indicated area on each page.
- Be precise and concise.
- When there is a box for an answer, **only the work in box provided will be graded.**
- You may use the blank page on the back for scratch work, but it will not be graded. Box numerical final answers.
- **The problems do not necessarily follow the order of increasing difficulty.** ***Avoid getting stuck on a problem.***
- Any algorithm covered in lecture can be used as a blackbox. Algorithms from homework need to be accompanied by a proof or justification as specified in the problem.
- You may assume that comparison of integers or real numbers, and addition, subtraction, multiplication and division of integers or real or complex numbers, require $O(1)$ time.
- There are warmup questions on the back page of the exam for while you wait.
- Good luck!

This page is deliberately blank. You may use it to report cheating incidents. Otherwise, we will not look at it.

Discussion Section

Which of these do you consider to be your primary discussion section(s)? Feel free to choose multiple, or to select the last option if you do not attend a section. **Please color the checkbox completely. Do not just tick or cross the boxes.**

- ☐ Arpita, Thursday 9 - 10 am, Dwinelle 223
- ☐ Avni, Thursday 10 - 11 am, Dwinelle 215
- ☐ Emaan, Thursday 10 - 11 am, Etcheverry 3107
- ☐ Lynn, Thursday 11 - 12 pm, Barrows 118
- ☐ Dee, Thursday 11 - 12 pm, Wheeler 30
- ☐ Max, Thursday 12 - 1 pm, Wheeler 220
- ☐ Sean, Thursday 1 - 2 pm, Etcheverry 3105
- ☐ Jiazheng, Thursday 1 - 2 pm, Barrows 175
- ☐ Neha, Thursday 2 - 3 pm, Dwinelle 242
- ☐ Julia, Thursday 2 - 3 pm, Etcheverry 3105
- ☐ Henry, Thursday 3 - 4 pm, Haviland 12
- ☐ Kedar, Thursday 3 - 4 pm, Barrows 104
- ☐ Ajay, Thursday 4 - 5 pm, Barrows 136
- ☐ Varun, Thursday 4 - 5 pm, Dwinelle 242
- ☐ Gillian, Friday 9 - 10 am, Dwinelle 79
- ☐ Hermish, Friday 10 - 11 am, Evans 9
- ☐ Vishnu, Friday 11 - 12 pm, Wheeler 222
- ☐ Carlo, Friday 11 - 12 pm, Dwinelle 109
- ☐ Tarun, Friday 12 - 1 pm, Hildebrand B56
- ☐ Noah, Friday 12 - 1 pm, Hildebrand B51
- ☐ Teddy, Friday 1 - 2 pm, Dwinelle 105
- ☐ Jialin, Friday 1 - 2 pm, Wheeler 30
- ☐ Claire, Friday 2 - 3 pm, Barrows 155
- ☐ Jierui, Friday 2 - 3 pm, Wheeler 202
- ☐ David, Friday 2 - 3 pm, Wheeler 130
- ☐ Nate, Friday 2 - 3 pm, Evans 9
- ☐ Rishi, Friday 3 - 4 pm, Dwinelle 243
- ☐ Tiffany, Friday 3 - 4 pm, Barrows 104
- ☐ Ida, Friday 3 - 4 pm, Dwinelle 109
- ☐ Don’t attend Section.

1 Asymptotics. (11 points.)

1. (**5 points, 1 point each.**) For each row, select the most specific asymptotic statement(s).

$f(n)$	$g(n)$	$f(n) = O(g(n))$	$f(n) = \Omega(g(n))$	$f(n) = \Theta(g(n))$
n^2	$1000n + \log n$	○	●	○
$2^{\log n}$	$2^{10 \log n}$	●	○	○
2^n	$2^{2^{\log n}}$	○	○	●
n^3	$n^{\log_2 7}$	○	●	○
$n!$	n^n	●	○	○

For 2^n vs $2^{2^{\log n}}$, we also accepted all the bubbles filled in, as the base of the log was not given.

2. (**3 points**) Consider the following statement: For $f(n), g(n) > 0$, if $\log f(n) = O(\log g(n))$, then $f(n) = O(g(n))$. Either prove the statement, or give a counterexample with justification.

Consider $f(n) = n^2$ and $g(n) = n$. Then $\log f(n) = 2n$ and $\log g(n) = n$. As big-O ignores constants, $f(n) = O(g(n))$. However, $f(n) \neq O(g(n))$ as the degree of $f(n)$ is strictly greater than that of $g(n)$

3. (**3 points**) Give a formal proof that $n = \Omega(\log n)$.

We have by L'Hôpital's Rule:

$$\lim_{n\to\infty} \frac{n}{\log n} = \lim_{n\to\infty} n = \infty$$

Therefore, by limit rules for asymptotics, $n = \Omega(\log n)$.

2 Recurrences. (13 points.)

Please give the tightest big O bound that you can.

1. $T(n) = 9T(n/3) + O(n^3)$

$O(n^3)$. There are $9^{\log_3 n} = n^2$ leaves and the work at the root is $O(n^3)$.

2. $T(n) = 9T(n/3) + O(n^2)$

$O(n^2 \log n)$. Here the work at every level is $O(n^2)$.

3. $T(n) = 4T(n-2) + O(2^n)$

$O(n2^n)$. There are 4^ℓ problems which take $2^{n-2\ell}$ time at depth i, for total work of 2^n per level. There are $n-2$ levels.

4. $T(n) = T(n/3) + T(2n/3) + O(n)$

   $O(n \log n)$. We love $O(n \log n)$.

   One can verify this using substitution. Here we make the constant in the $O(n)$ explicit:

$$T(n) \leq T(n/3) + T(2n/3) + cn.$$

   Then we show inductively that there is a c' where $T(n) \leq c' n \log n$ as follows.

$$\begin{aligned} T(n) &\leq c'(n/3 \log(n/3)) + c'(2n/3)\log(2n/3) + cn \\ &\leq c'(n/3 \log(2n/3)) + c'(2n/3)\log(2n/3) + cn \\ &\leq c' n \log(2n/3) + cn \\ &\leq c' n \log(n) - c' n \log(3/2) + cn \end{aligned}$$

   Choosing $c' > c/\log 3/2$ will ensure we satisfy the equation above.

5. $T(n) = 3T(n^{1/3}) + O(\log n)$

   $O(\log n \log\log n)$.

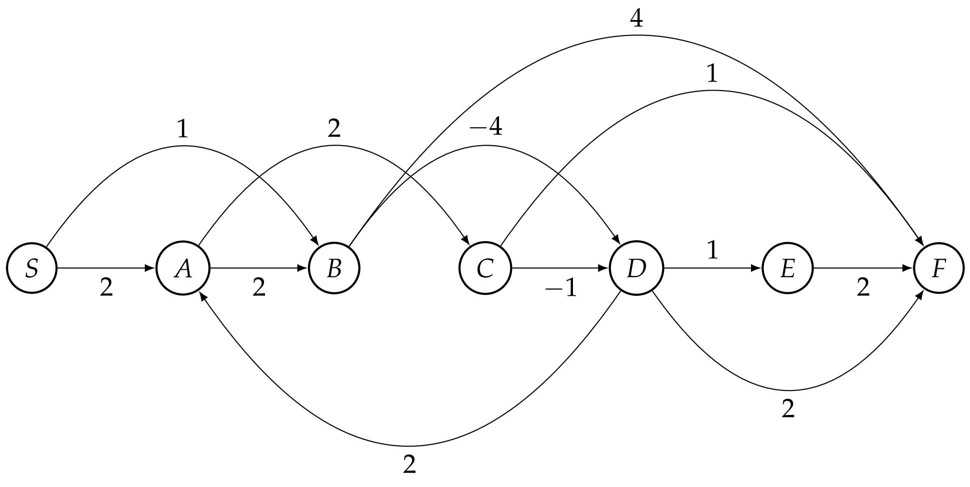
   Substitute $x = \log n$

   $T(x) = 3T(x/3) + O(x)$.

   Here, it is easy to see this is $O(x \log x)$.

3 Update, here, there, not quite everywhere. (6 points.)

Consider the graph below with the edge weights as indicated.



1. Give the shortest path distances to each vertex from S. (Be sure to notice all edges are directed from left to right, except the arc from D to A which is directed from right to left.)

S:	0	A:	-1	B:	1	C:	1
D:	-3	E:	-2	F:	-1		

2. Give a sequence of 6 edge updates that starting from $d(S) = 0$, and $d(X) = \infty$ for all $X \neq S$, produces a valid set of shortest path distances. (We give you the first one.)

Update 1:	(S, B)	Update 2:	(B, D)
Update 3:	(D, A)	Update 4:	(D, E)
Update 5:	(D, F)	Update 6:	(A, C)

4 Short answers and True/False. (38 points. 2 points each part.)

1. Let ω_n be the nth primitive root of unity in the complex numbers. We let u^* be the vector where each complex number is replaced by its complex conjugate.

   Recall for a complex number $a = re^{i\theta}$, its conjugate is $a^* = re^{-i\theta}$. Notice that $aa^* = r^2e^0 = r^2$.

   Let $u = (1, \omega_n, \omega_n^2, \ldots, \omega_n^{n-1})$ and $v = (1, \omega_n^2, \omega_n^4, \ldots, \omega_n^{2(n-1)})$.

   (a) What is $u \cdot u^*$? (Recall $x \cdot y$ for vectors x and y is $\sum_i x_i y_i$.)

   n. Each complex number has magnitude 1, and the magnitude squared for a complex number by multiplying by its conjugate complex number.

   (b) What is $u \cdot v^*$? (Recall u^* is the vector of conjugates of the elements of u)

   0. The dot product is
   $$\sum_i (\omega_n^3)^i.$$
   This can be seen to be zero (as in your homework) by using the formula for a geometric sum
   $$\sum_i^{n-1} r^i = \frac{1 - r^n}{/} 1 - r$$
   and we use $1 - (\omega_n^3)^n = 1 - (\omega_n^n)^3 = 1 - 1^3 = 0$.

2. Recall that ω_n is a primitive nth root of unity. That is, $\omega_n = e^{2\pi i/n}$.

   Let $n = 2^k$. What is the smallest $t > 0$ where $(\omega_n^4)^t = 1$?

   (Answer is an expression possibly using n and k.)

   2^{k-2} as $((\omega_n)^4)^{2^k/4} = 1$. And XXX it is primitive since...

3. We wish to evaluate $A(x) = a_0 + a_1x + a_2x^2 + a_3x^3$ for $x \in \{1, 2, 3, 4\}$ as follows. (Fill in the blanks so that the matrix computes the evaluations of $A(x)$)

$$\begin{bmatrix} \_\_\_ & \_\_\_ & \_\_\_ & \_\_\_ \\ \_\_\_ & \_\_\_ & \_\_\_ & \_\_\_ \\ \_\_\_ & \_\_\_ & \_\_\_ & \_\_\_ \\ \_\_\_ & \_\_\_ & \_\_\_ & \_\_\_ \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix} = \begin{bmatrix} A(1) \\ A(2) \\ A(3) \\ A(4) \end{bmatrix}$$

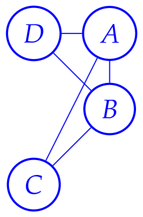
$$\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 2 & 4 & 8 \\ 1 & 3 & 9 & 27 \\ 1 & 4 & 16 & 64 \end{bmatrix}$$

4. For a DFS postordering for a directed graph, $G = (V, E)$, for every edge $(u, v) \in E$, $post(u) > post(v)$.

   False. A back edge has $post(u) < post(v)$ as it points to an ancestor v where explore(v) completes after the explore(u) and thus has a larger post order number.

5. For an undirected graph, if there is a vertex of degree at least 3 in a DFS tree, removing that vertex disconnects the graph.

   False. Consider the graph, start depth first search from A, and explore B next.



6. For every graph with a Hamiltonian cycle (a simple cycle which contains every vertex), depth first search could produce a tree of depth $n - 1$.

   True. Order the edges in the Hamiltonian cycle first in each vertice's adjacency list where DFS explores the edges in order of adjacency list. The cycle will be explored first.

7. For a DAG with a single source s (a vertex with no incoming edges), all vertices are reachable from s.

   True. You did this on homework! Topologically sort the vertices. Inductively, each prefix is reachable from the source. The base case is that the source is reachable from the source, for each subsequent vertex its incoming edge comes from previously reachable (from the source) vertices.

8. For any depth first search of a DAG, the vertex with the lowest post-order number is a sink.

   True.

   The inverse post-order number is a topological sort, thus the lowest post ordered number vertex is a sink.

9. If there is a unique minimum weight edge in an undirected connected weighted graph, then it must be in every minimum spanning tree.

   True. If not, there is a cut that contains it and another edge, and exchanging the two edges would decrease the cost of the supposedly optimal tree.

10. If a weighted undirected connected graph remains connected after removing all edges with weight greater than w, then every edge in any MST has weight at most w.

    True. Any edge e of weight greater than w in a supposedly minimum spanning tree T can be removed and the connected components of $T - e$ correspond to a cut which must at least one edge e' of weight at most w, replacing e with e' produces a smaller weight tree. This is a contradiction.

11. For every weighted graph $G = (V, E)$ and cut $(S, V - S)$, there is an MST which contains all minimum weight edges across that cut.

    False. Consider a square where the top and bottom edge are 1, and the left and right edges are 2.

12. Given a graph, $G = (\{1, \ldots, n\}, E)$, where $n > 3$, and E contains the edges $(1, 2)$, $(2, 3)$ and $(1, 3)$ with edge weights 1, 1, and 2 respectively:

    (a) Which, if any, of the edges $(1, 2), (2, 3), (1, 3)$ must be in every MST?
    (Give the edge(s) or state None.)

    None. The graph could be connected by other edges.

    (b) Which, if any, of the edges $(1, 2), (2, 3), (1, 3)$ cannot be in any MST?
    (Give the edge(s) or state None.)

    $(1, 3)$. The heaviest edge on a cycle is not on a cut, since in any tree at least one lighter edge on the cyle is not in the tree, and one can replace the heavy edge with the light edge to produce a cheaper tree.

13. Recall Bellman-Ford on a graph $G = (V, E)$ from a source s initializes $d(s)$ to 0 and $d(v) = \infty$ for all other vertices v. Then, it updates all the edges $|V| - 1$ times.

    (a) If there are at most k negative length edges in G, then updating all edges k times is sufficient to compute all shortest path distances from s.

    False. Consider a path of length $2k$ with no negative arcs, the distance label of the end of the path may remain ∞.

    (b) If every shortest path from s to any vertex uses at most k edges,then updating all edges k times is sufficient to compute shortest path distances from s.

    True. The induction for Bellman-Ford says after t iterations the distance label to each vertex is at most the distance of any path that uses at most t edges.

14. Consider an undirected graph G with vertices $\{A, B, C, D, E, F, H\}$, where a breadth first search computes the following distances:

$$[A : 0], [B : 1], [C : 1], [D : 1], [E : 2], [F : 2], [H : 3].$$

    (a) There cannot be an edge (B, H) in G.

    True. Otherwise $d(A, F) \leq d(A, B) + 1 = 2$.

    (b) Identify an edge that must exist G.

    (A, B) or (A, C) or (A, D). Any distance 1 vertex must be directly connected to the source A.

    (c) Identify the smallest set of vertices that is guaranteed to be a vertex cut. (Recall from homework, a vertex cut is a set of vertices whose removal leaves a disconnected graph.)

    E, F. There is no edge from A to H when removing these vertices as otherwise H's distance label would be 1. Similarly, there are no edges from B, C, D to H otherwise H's distance would be 2

5 Polynomial Multiplication/Applications. (8 points. 2/2/4 points by part.)

1. Given that $A(x) = a_0 + a_1x + a_2x^2$ and $B(x) = b_0 + b_1x + b_2x^2$, what is the coefficient of x^2 in $A(x) \times B(x)$? (In terms of a_0, a_1, a_2, b_0, b_1 and b_2.)

   $a_0b_2 + a_1b_1 + a_2b_0$. The terms come from $(a_0 + a_1x + a_2x^2 + \ldots)(b_0 + b_1x + b_2x^2 + \ldots)$.

2. Given $A(x) = a_0 + a_1x + a_2x^2 + \ldots a_nx^n$ and $B(x) = b_0 + b_1x + b_2x^2 \ldots a_nx^n$, what is the coefficient of x^k in $A(x) \times B(x)$? (In terms of $a_0, a_1, \ldots a_n$ and $b_0, \ldots, b_n$.)

   $\sum_{i=0}^{k} a_ib_{k-i}$. The terms in the sum come from products of the form $a_ix^k + b_{k-i}x^{k-i} = a_ib_{k-i}x^k$.

3. Let A and B be independent random variables which take on values in $[0, \ldots, n-1]$, where $Pr[A = k] = a_k$ and $Pr[B = k] = b_k$.

   Briefly describe an $O(n \log n)$ time algorithm that, given $a_0, \ldots, a_{n-1}$ and $b_0, \ldots, b_{n-1}$, computes the probability mass function for $A + B$, i.e., computes $Pr[A + B = k]$ for all relevant k.

   (Hint: consider all the values of A and B where $A + B = k$.)

   Note that by the total probability rule $\Pr(A + B = k) = \sum_{i=0}^{k} \Pr(A = i, B = k - i) = \sum_{i=0}^{k} a_ib_{k-i}$.

   Now let $P_A(x) = a_0 + a_1x + \ldots + a_{n-1}x^{n-1}$, and $P_B(x) = b_0 + b_1x + \ldots + b_{n-1}x^{n-1}$, then see that the coefficient for x^k in $P_A \cdot P_B$ is $\sum_{i=0}^{k} a_ib_{k-i}$.

   Compute the polynomial multiplication using FFT, then return $\Pr(A + B = k)$ as the x^k coefficient in the result. This takes $\mathcal{O}(n \log n)$ time as both polynomials have degree at most $n - 1$.

6 Divide and Conquer. (12 points. 6/6 per part.)

1. Given k sorted lists, each of length n, give an algorithm to produce a single sorted list containing the elements of all the lists. The runtime should be $O(kn \log k)$.

   Possible accepted solutions:

   (a) Call the algorithm recursively on the first $k/2$ sorted lists, then on the second $k/2$ sorted lists. Then merge them together (this takes $kl/2$ time). The recurrence is $T(k) = 2T(k/2) + \mathcal{O}(kn)$. This gives time $kn + 2(k/2)n + \ldots + 2^{\log k}(k/2^{\log k})n = \mathcal{O}(kn \log k)$.

   (b) Do the k-way merge all at once. Keep a pointer to the start of each array. Add the smallest element in each array to a heap.

   Repeatedly pull the smallest item from the heap and increment the pointer of the array it came from. Then add the destination of the pointer to the heap. Do this until the heap is empty and all pointers point past their arrays.

   This is kn heap operations (delete, insert) on a heap of size at most k, giving time $\mathcal{O}(kn \log k)$.

   (c) Similar to the above, put the smallest elements from each array into a new k-length array A and sort them. Keep a pointer to the start of each array.

   Repeatedly pull the smallest element from A, increment the pointer it came from. Use binary search to insert the destination of the pointer into the right place into A. Do this until the array is empty and all pointers point past their arrays.

   It is not obvious how to implement this, as you need an array with constant-time seek and insert operations. If you use an amortized data-structure, you will get $\mathcal{O}(kn \log k)$ as well.

It is not enough to merge the first two lists, then merge the result with the third, then the result with the fourth, etc. This would give a $k^2 \log n$ runtime.

2. In the MultipleSelect problem that given a set of n items and a set r of select requests $k_1, \ldots, k_r$.

   Consider the algorithm of finding a median element (using deterministic median finding) and recursively computing MultipleSelect on each half with the appropriate requests.

   Argue that this algorithm takes at most $O(n \log r)$ time.

   (Note: if there are no requests in one of the halves, one does not need to make a recursive call on that half.)

   The algorithm as given in the problem computes the median (in $\mathcal{O}(n)$ time), then partitions the elements around the median, and the requests $k_1, \ldots, k_r$ around $n/2$.

   After $\log(2r) = \mathcal{O}(\log r)$ recursive calls, we have partitioned the original array into $2r$ pieces with $n/2r$ elements each. As there are only r requests, at most half of these pieces have a relevant request. The algorithm will only need to recurse into these pieces, so we have essentially halved the size of the problem.

   At an arbitrary level i, the amount of work done is $2^i \cdot n/2^i = n$ (to split each of the 2^i partitions around their medians). The amount of work done is $\mathcal{O}(n)$ at each level, and there are $\mathcal{O}(\log r)$ levels, so the total amount of work to get to this point is $\mathcal{O}(n \log r)$.

   This gives us the recurrence $T(n) = T(n/2) + \mathcal{O}(n \log r)$. This expands as a series $\log r(n + n/2 + n/4 + \ldots) = \mathcal{O}(n \log r)$.

   Formal proof: The above argument is fine and was given full credit, the following is how we might do it rigorously.

   Let $T(n,r)$ be the time to do r selects on n items. We know that $T(n,0) = 0$ for all n. We show by induction that $T(n,r) \le c' \cdot n \log r$ for some $c' > 0$.

   After $\log(2r)$ recursive calls, let r_i count the number of requests remaining in the ith partition. Then $\sum_{i=1}^{2r} r_i = r$. As the r_i are non-negative integers, this means that at least n of them must be zero.

   Assume $T(n,r) \le c' \cdot n \log r$ for all integers smaller than n. We get the recurrence $T(n,r) = \sum_{i=1}^{2r} T(n/2r, r_i) + \mathcal{O}(n \log r)$. Replace $\mathcal{O}(n \log r)$ with $\le d \cdot n \log r$ and use the fact that at least n of the r_i are zero and $T(n,0) = 0$:

$$\begin{aligned}
T(n,r) &\le \sum_{i=1}^{2r} T\left(\frac{n}{2r}, r_i\right) + d \cdot n \log r \\
&= \sum_{i:\, r_i > 0} T\left(\frac{n}{2r}, r_i\right) + d \cdot n \log r \\
&\le \sum_{i:\, r_i > 0} c \cdot \frac{n}{2r} \log r_i + d \cdot n \log r \\
&\le \frac{cn}{2r} r \log r + d \cdot n \log r \\
&\le \left(\frac{c}{2} + d\right) n \log r
\end{aligned}$$

   We need to pick $c \ge \frac{c}{2} + d$, so $c \ge 2d$.

7 Strongly connected components. (6 points.)

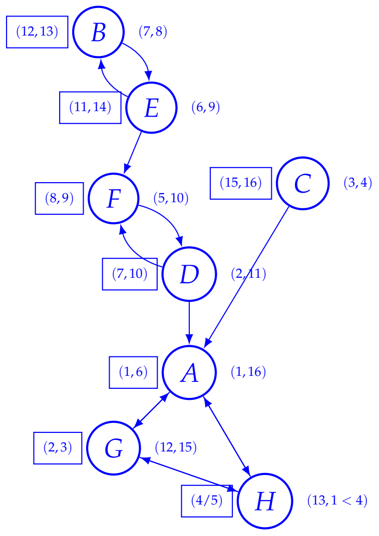
Recall, the strongly connected components algorithm on a graph G proceeds by running depth first search on the reverse graph, G^R, and then runs depth first search on G using inverse post order number.

Let G have vertices a, b, c, d, e, f, g, h. The table below illustrates the post order numbering of the first run, and the post order numbering of the second run on G, using the inverse post ordering from the first run.

	a	b	c	d	e	f	g	h
$post(G^R)$	16	8	4	11	9	10	15	14
$post(G)$	6	13	16	10	14	9	4	5

1. What are the strongly connected components (SCCs) of the graph?
2. What is the topological sort of the strongly connected components of the graph?

Here is a graph which yields the post orderings above. The boxed ones are for the second run.



1. The components are $\{A, G, H\}, \{B, E\}, \{C\}, \{D, F\}$.

   One gets these by noticing that A has the highest post order number in the first one, and noting that G, H are explored by A since their post order numbers are lower in the second run. One can then go to D and see that F is explored and so on.

2. The topological sort is $\{C\}, \{B, E\}, \{D, F\}, \{A, G, H\}$.

   One can get a topological sort of the graph, by noticing that the highest post order number must be in a source component in the final run of the DFS. This is true again after removing that source. Thus, the highest post order number is in $\{C\}$, the next is E which is in $\{B, E\}$, next is $\{D, F\}$ and so on.

8 Shortest Path: remove an edge. (10 points)

Consider an undirected graph $G = (V, E)$ with non-negative edge weights. You are also given a shortest path labelling, $d(\cdot)$, and a corresponding shortest path tree $\mathcal{T}$.

Consider removing an edge (u, v) in $\mathcal{T}$. Let S be the set of v's descendants in $\mathcal{T}$ and $E(S)$ be the set of edges where at least one of the endpoints is in S.

Give an $O(|E(S)| \log |S|)$ algorithm to update the distance labels in $d(\cdot)$ to be the shortest path distances in the resulting graph.

One can remove the parent edge and compute the descendants, S, of e in $O(|S|)$ time using DFS on the tree edges.

Notice the shortest path label of any vertex not in S is correct as the previous shortest path remains undisturbed by this edge removal and we have not created any new paths. Let's say these vertices are $D = V - S$.

Thus, essentially we want to modify dijkstra's so that we only update distances that could possibly change in S.

So, we set $d(u) = \infty$ for each vertex u in S and then update all vertices that have a path through D to be the min of the paths through D. To do this, we assign d(u) to be the min of $d(x) + d(x, u)$ where x is a vertex in D and u is a vertex in S. We then add all vertices u in S to the priority queue, with priority of $d(u)$.

We then process these vertices in order of $d(\cdot)$ and update for all edges with both endpoints in S.

The correctness follows since the vertices in D have the correct label and we always start by pulling a vertex u whose path go through D. Through induction on subsequent steps since paths can never get smaller (because the edges are non-negative), we correctly compute new paths either go directly through D, or paths that have been previously computed so go through D and vertices in S.

The initialization step will take $E(S)$ time because we can scan through all edges in $E(S)$ and update $d(\cdot)$ accordingly. Then for dijkstra's since we only run it on vertices in S, we get the runtime to be $O((E(S)) + |S|) \log(|S|))$. The dijkstra's step dominates the initialization step and since $E(S) \geq |S|$, we can drop $|S| log(|S|)$ and so the overall runtime is $O(E(S) log(|S|))$.

9 Points on a Line. (6 points.)

You are given a list $L = [x_1, x_2, \dots, x_n]$ of n points on the real line where n is even and $x_1 < x_2 < \dots < x_n$. Design a greedy algorithm that partitions L into $n/2$ pairs (a_i, b_i), $i = 1, 2, \dots, n/2$ to minimize:

$$\sum_{i=1}^{n/2} |a_i - b_i|$$

Algorithm is to repeatedly match the two smallest points.

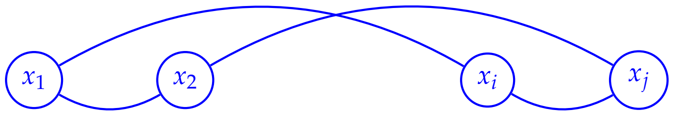
This is correct as if the first pair (x_1, x_2) are not in a matching, than the points are are matched (x_1, x_i) and (x_2, x_j) where $x_i, x_j > x_2$.

Assuming $x_j > x_i$, we have the cost of the two pairs to be

$$(x_i - x_1) + (x_j - x_2) = (x_i - x_2) + (x_2 - x_1) + (x_j - x_i) + (x_i - x_2) = (x_j - x_i) + (x_2 - x_1) + 2(x_i - x_2).$$

Matching (x_1, x_2) and (x_j, x_i) gives $(x_2 - x_1) + (x_j - x_i)$ which is no bigger as $2(x_i - x_2) \geq 0$.

Here is a figure:



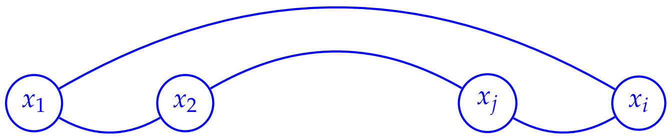
We exchanged the upper edges with the lower ones.

If $x_i > x_j$, we have the cost of the two pairs as:

$$(x_i - x_1) + (x_j - x_2) = (x_i - x_j) + (x_j - x_2) + (x_2 - x_1) + (x_j - x_2) = (x_i - x_j) + (x_2 - x_1) + 2(x_j - x_2),$$

which again is at least as large as $(x_2 - x_1) + (x_i - x_j)$ as

$$2(x_j - x_2).$$



Again, the exchange is the top edges with the bottom ones.

Thus, there is an optimal solution with the first two points paired.

Furthermore, then one can apply induction that the algorithm gives a correct solution on the $n - 2$ points to conclude the argument.