Слайд 2:  
Слайд 3:  
Слайд 4:  
Слайд 5:

Тім ввів таке поняття як minrun – найменший розмір відсортованих підмасивів(тобто run’ів) для подальшого злиття. Причому експериментальним шляхом визначивши, що найкраща асимптотика досягається при розмірі від 32 до 64. **Справа в тому, що при малому minrun’і виходить дуже багато шматочків, які необхідно об'єднати злиттям.**

**А при великому minrun’і сортування вставками, яка сортує кожний run, буде надто довго працювати. Так як у кращому випадку асимпотика дорівнює n^2. Також було визначено, що в ідеалі для збалансованих злиттів кількість run’ів має бути степеня двійки, а якщо розмір кожного run`а повинен дорівнювати minrun’у або інколи більше, то необхідно визначити такий minrun з інтервалу від 32 до 64, щоб визначення розміру всього масиву на minrun вийшла степінь двійки.** **Ну чи трохи менше, щоб незбалансованих злиттів було небагато.** При цьому якщо розмір масиву менше 64, TimSort перетворюється на просте сортування вставками.  
Слайд 6-7:

За задумом необхідно розбити масив на відсортовані підпослідовності. Це робиться в такий спосіб. Ставиться покажчик початку нашого масиву. З поточного елемента починається пошук run'а. За визначенням поточний та наступний елементи однозначно входять у run, який буде впорядковано аналогічно цим першим двом елементам. Або за нестрогим зростанням, або за спаданням. **Таким чином, кожен наступний елемент перевіряється за цією умовою і при його виконанні елемент додається в run. Навіть якщо вже досягнуто розміру мінран. Але якщо ран все ще не набраний, а трапляється елемент, що порушує нашу монотонність, то він додається в run за допомогою бінарного пошуку. Після чого триває набір run'а поки одночасно**

**не буде досягнутий розмір minrun'a і наступні елементи перестануть підходити під умови. (виходить, що ран завжди більше або дорівнює minrun’у) Отримуємо що наші нові елементи додаються сортуванням вставками. Наступні рани будуються аналогічно.**  
Слайд 8:

Після того як ми отримали run. Він додається до стеку у вигляді пари чисел з індексу початку run’а та його довжини. (пари порівнюються довжиною) Однак стек будується за певними правилами, нехай останніми в стек були додані елементи X, Y і Z. Тоді важливо дотримуватися наступних умов: Y повинен бути строго більше Z, a X строго більше суми Y і Z. **Відповідно якщо в стеку два елементи, то важливо виконання лише першої умови. Якщо ж одна з умов порушиться, то застосовується сортування злиттям Y з меншим з X і Z. У випадку двох елементів вони просто зливаються один з одним. Якщо після злиття в стеку залишилося більше одного елемента, то знову перевіряється виконання цих умов і при їх порушенні алгоритм повторюється ще раз. Таким чином (досягається інваріант стан стека і при цьому) гарантується зростання швидше зростання чисел Фібоначчі.**  
Слайд 9:  
Для злиття у виділену додаткову пам'ять копіюється менший масив. Якщо менший виявився зліва, то покажчики ставиться на перші елементи і біжать по масиву зліва

праворуч як із звичайному сортуванні злиттям. Якщо менше справа, то покажчики навпаки біжать справа ліворуч. Таким чином по порядку порівнюються елементи масиву і копіюється менше (мінімум) з них, якщо покажчики йдуть зліва направо. І більше (максимум, копіюється, якщо справа наліво) у протилежному випадку. **Коли покажчик доходить до кінця більшого з масивів, то частина меншого, що залишилася, повністю копіюється і прибудовується відповідно зліва або праворуч.**

**У випадку з досягненням кінця меншого масиву частина більшого, що залишилася, вже стоїть на своєму місці.**  
Слайд 10:

Однак ця ідея не застосовується у чистому вигляді Тім Петрс запропонував використовувати модернізацію, яка отримала назву галоповий пошук. Суть її в наступному: тому що ймовірність знаходження меншого елементів більша в

початку масиву. Спочатку перші 7 елементів порівнюються

лінійно і якщо всі вони були взяті з одного масиву, то далі алгоритм переходить в режим галопу, тобто покажчик у цьому масиві зміщується спочатку на один елемент, потім на 2, потім на 4, на 8, 16 і так далі поки елемент перестане підходити або доки не закінчиться масив. **Далі якщо такий елемент знайшовся, бінарним пошуком на пройденому галопом відрізку знаходиться останній відповідний елемент. Після цього копіюється відразу шматок масиву від початку галопу до знайденого елемента і алгоритм повертається в режим лінійного пошуку.**

**Якщо ж був досягнутий кінець масиву, то копіюються відразу вся його частина. Особливо ефективний цей підхід при порівнянні складних об'єктів.**  
Слайд 11:  
Слайд 12:  
Слайд 13: