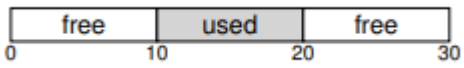
17

Free-Space Management

Bu bölümde, herhangi bir bellek yönetim sisteminin temel bir yönünü tartışmak için belleği sanallaştırma tartışmamızdan küçük bir sapma yapacağız. Bir malloc kitaplığı (bir işlemin yığınının sayfalarını yönetme) veya işletim sisteminin kendisi (bir işlemin adres alanının bölümlerini yönetme) olabilir. Spesifik olarak, **boş alan yönetimi (free-space management)** çevreleyen konuları tartışacağız.

Öncelikle sorunu daha spesifik hale getirelim. Boş alanı yönetmek kesinlikle kolay olabilir,**bellek adresleme (paging)** kavramını tartışdığımızda görebileceğimiz gibi.Yönettiğiniz alan sabit boyutlu birimlere bölündüğünde kolaydır; bu durumda, sabit boyutlu birimlerin bir listesini tutmanız yeterlidir; ne zaman istemci bunlardan birini ister, ilk girişi döndürür.

Boş alan yönetiminin daha zor (ve ilginç) hale geldiği durum, yönettiğiniz boş alanın değişken boyutlu birimlerden oluşmasıdır.Bu kullanıcı düzeyinde bir bellek ayırma kitaplığında (malloc( ) ve free( ) gibi) ve fiziksel belleği yöneten bir işletim sisteminde sanal belleği somutlaştırmak için **parçalara ayırma (segmantation)** kullanılan durumlardır.Her iki durumda da ,sorun var olan **harici parçalanma (external fragmentation)** olarak bilinir: Boş alan farklı boyutlarda küçük parçalar bölünür ve böylece parçalanır; toplam boş alan miktarı isteğin boyutunu aşsa bile isteği karşılayabilecek tek bir bitişik alan olmadığı için sonraki istekler başarısız olabilir.



Şekil bu sorunun bir örneğini göstermektedir. Bu durumda toplam kullanılabilir boş alan 20 bayttır ; ne yazık ki, her biri 10'luk iki parçaya bölünmüş durumda. Sonuç olarak, 20 bayt boş olsa bile 15 baytlık bir istek başarısız olur.Böylece bu bölümde ele alınan soruna ulaşmış oluyoruz.

SORUNUN MERKEZİ:BOŞ ALAN NASIL YÖNETİLİR

Değişken boyutlu istekler karşılanırken, boş alan nasıl yönetilmelidir? Parçalanmayı en aza indirmek için hangi stratejiler kullanılabilir?Alternatif yaklaşımların zaman ve mekan genel giderleri nelerdir?

17.1 Varsayımlar

Bu tartışmanın çoğu, kullanıcı düzeyinde bellek ayırma kitaplıklarında bulunan ayırıcıların harika geçmişine odaklanacaktır. Wilson'ın mükemmel anketinden [W+95] yararlanıyoruz, ancak ilgili okuyucuları daha fazla ayrıntı için kaynak belgeye gitmeye teşvik ediyoruz.

`malloc()` ve `free()` tarafından sağlananlar gibi temel bir arayüz varsayıyoruz.Özellikle, `void *malloc (size_t size)` tek bir parametre alır, `size` , uygulama tarafından talep edilen bayt sayısı; o boyuttaki (veya daha büyük) bir bölgeye bir işaretçiyi (belirli bir türde olmayan veya C dilinde bir **boş işaretçi(void pointer))** geri verir. Tamamlayıcı yordam `void free(void * ptr)`, bir işaretçi alır ve karşılık gelen öbeği serbest bırakır.Arayüzün mantıksal anlamına dikkat edin.Kullanıcı,boşluk,kitaplığa boyutu hakkında bilgi vermez; bu nedenle, kitaplık, kendisine yalnızca bir işaretçi verildiğinde, bir bellek yığınının ne kadar büyük olduğunu anlayabilmelidir. Bunu nasıl yapacağımızı biraz sonraki bölümde tartışacağız.

Bu kitaplığın yönettiği alan, tarihsel olarak `heap` diye bilinir, ve yığındaki boş alanı yönetmek için kullanılan genel veri yapısı, bir tür **boş listedir(free list).** Bu yapı, belleğin yönetilen bölgesindeki tüm boş alan yığınlarına referanslar içerir. Tabii ki, bu veri yapısının kendi başına bir liste olması gerekmez, ancak boş alanı izlemek için sadece bir tür veri yapısı olması gerekir.

Ayrıca, yukarıda açıklandığı gibi, öncelikli olarak **harici parçalanma(external fragmentation)** ile ilgilendiğimizi varsayıyoruz. Tahsis edenler, elbette **dahili parçalanma(internal fragmentation)** sorunu da yaşayabilirler; bir ayırıcı, talep edilenden daha büyük bellek yığınları dağıtırsa, böyle bir yığındaki sorulmamış (ve dolayısıyla kullanılmayan) herhangi bir alan, dahili parçalanma olarak kabul edilir (çünkü atık, tahsis edilen birimin içinde gerçekleşir) ve alan israfına başka bir örnektir. Bununla birlikte, basitlik adına ve iki tür parçalanmadan daha ilginç olduğu için, çoğunlukla harici parçalanmaya odaklanacağız.

Ayrıca, bellek istemciye dağıtıldıktan sonra, bellekte başka bir konuma taşınamayacağını da varsayacağız. Örneğin, bir program `malloc()`'u çağırırsa ve yığın içindeki bir boşluğa bir işaretçi verilirse, bu bellek bölgesi, program karşılık gelen bir `free()` çağrısı yoluyla onu döndürene kadar esasen programa "aittir" (ve kitaplık tarafından taşınamaz).Böylece ,boş alanın **sıkıştırılması (compaction)**mümkün değildir,

parçalanmayla mücadelede yararlı olacaktır. Bununla birlikte, sıkıştırma, işletim sisteminde bölümlendirmeyi uygularken **parçalarına ayırma (segmentation)** ile başa çıkmak için kullanılabilir (bölümleme ile ilgili bölümde tartışıldığı gibi).

Son olarak, ayırıcının bitişik bir bayt bölgesini yönettiğini varsayacağız. Bazı durumlarda, bir tahsisatçı o bölgenin büyümesini isteyebilir; örneğin, kullanıcı düzeyinde bir bellek ayırma kitaplığı, alanı dolduğunda yığını büyütmek için çekirdeği arayabilir (`sbrk` gibi bir sistem çağrısı yoluyla). Ancak, basit olması için, bölgenin ömrü boyunca tek bir sabit boyutta olduğunu varsayacağız.

17.2 Düşük seviyeli mekanizmalar

Bazı ilke ayrıntılarına girmeden önce, çoğu tahsisatçıda kullanılan bazı ortak mekanizmaları ele alacağız. İlk olarak, herhangi bir ayırıcıda yaygın olarak kullanılan teknikler olan bölme ve birleştirmenin temellerini tartışacağız. İkinci olarak, tahsis edilen bölgelerin boyutunun nasıl hızlı ve nispeten kolay bir şekilde takip edilebileceğini göstereceğiz. Son olarak, hangi bölgenin boş olup olmadığını takip etmek için boş alan içinde nasıl basit bir liste oluşturacağımızı tartışacağız.

Bölünme ve Birleştirme

Boş bir liste, yığında hala kalan boş alanı tanımlayan bir dizi öğe içerir. Böylece, aşağıdaki 30 baytlık yığını varsayalım:

Bu yığın için boş listede iki öğe olacaktır. Bir giriş, ilk 10 baytlık boş bölümü (0-9 bayt) ve bir giriş diğer boş bölümü (20-29 bayt) açıklar:

Yukarıda açıklandığı gibi, 10 bayttan büyük herhangi bir istek başarısız olur (NULL döndürür); o boyutta tek bir bitişik bellek parçası yok. Tam olarak bu boyuta (10 bayt) yönelik bir istek, boş parçalardan herhangi biri tarafından kolayca karşılanabilir. Ancak istek 10 bayttan `küçük` bir şey içinse ne olur?

Yalnızca tek bir bayt bellek talebimiz olduğunu varsayalım. Bu durumda, **ayırıcı bölme(spilitting)** olarak bilinen bir eylemi gerçekleştirecektir:

isteği karşılayabilecek ve onu ikiye bölebilecek boş bir bellek parçası bulacaktır. İlk yığın arayan kişiye geri dönecektir; ikinci yığın listede kalacaktır. Bu nedenle, yukarıdaki örneğimizde, 1 baytlık bir talepte bulunulursa ve ayırıcı, talebi karşılamak için listedeki iki öğeden ikincisini kullanmaya karar verirse, malloc() çağrısı 20 döndürür (1 baytlık ayrılmış bölgenin adresi) ve liste şöyle görünür:

Resimde, listenin temelde bozulmadan kaldığını görebilirsiniz; tek değişiklik, serbest bölgenin artık 20 yerine 21'den başlaması ve bu serbest bölgenin uzunluğunun artık sadece 9^3 olmasıdır. Bu nedenle, istekler herhangi bir belirli boş yığının boyutundan daha küçük olduğunda ayırma genellikle ayırıcılarda kullanılır.

Pek çok ayırıcıda bulunan doğal bir mekanizma, boş alanın **birleştirilmesi(coalescing)** olarak bilinir. Örneğimizi bir kez daha ele alalım (ücretsiz 10 bayt, kullanılmış 10 bayt ve başka bir boş 10 bayt).

Bu (küçük) yığın göz önüne alındığında, bir uygulama boş(10) öğesini çağırdığında ve böylece yığının ortasındaki boşluğu döndürdüğünde ne olur? Bu boş alanı çok fazla düşünmeden listemize geri eklersek, şöyle görünen bir liste elde edebiliriz:

Soruna dikkat edin: yığının tamamı artık özgürken, görünüşe göre her biri 10 baytlık üç parçaya bölünmüştür. Bu nedenle, bir kullanıcı 20 bayt isterse, basit bir liste geçişi bu kadar boş bir öbek bulamayacak ve başarısızlıkla sonuçlanacaktır.

Ayırıcıların bu sorunu önlemek için yaptığı şey, bir yığın bellek serbest bırakıldığında boş alanı birleştirmektir. Fikir basit: bellekte boş bir yığın döndürürken, iade ettiğiniz yığının adreslerine ve yakındaki boş alan parçalarına dikkatlice bakın; yeni serbest kalan alan bir (veya bu örnekte olduğu gibi iki) mevcut boş parçanın hemen yanında bulunuyorsa, bunları daha büyük tek bir boş yığın halinde birleştirin. Böylece, birleşme ile nihai listemiz şöyle görünmelidir.

Gerçekten de, herhangi bir ayırma yapılmadan önce yığın listesi ilk bakışta böyle görünüyordu. Birleştirme ile bir ayırıcı, uygulama için geniş boş uzantıların kullanılabilir olmasını daha iyi sağlayabilir.

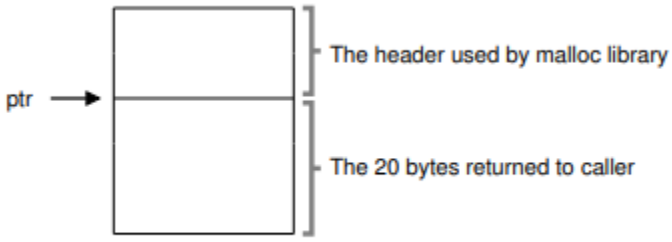


Figure 17.1: **An Allocated Region Plus Header**

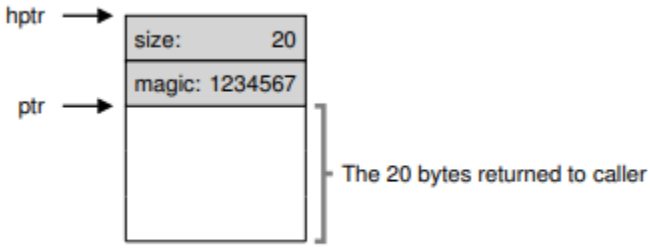


Figure 17.2: **Specific Contents Of The Header**

Tahsis Edilen Bölgelerin Büyüklüğünün Takibi

`free(void *ptr)` arabiriminin bir boyut parametresi almadığını fark etmiş olabilirsiniz; bu nedenle, bir işaretçi verildiğinde, malloc kitaplığının serbest bırakılan bellek bölgesinin boyutunu hızlı bir şekilde belirleyebileceği ve böylece alanı tekrar serbest listeye dahil edebileceği varsayılır.

Bu görevi gerçekleştirmek için, çoğu ayırıcı, genellikle dağıtılan bellek öbeğinden hemen önce, bellekte tutulan bir **başlık(header)** bloğunda biraz fazladan bilgi depolar. Tekrar bir örneğe bakalım (Şekil 17.1). Bu örnekte, `ptr` ile gösterilen 20 baytlık tahsis edilmiş bir bloğu inceliyoruz; `malloc()` adlı kullanıcının ve sonuçları `ptr`'de sakladığını düşünün. `ptr =malloc(20);` gibi.

Başlık, minimum olarak tahsis edilen bölgenin boyutunu içerir (bu durumda 20); ayrıca serbest ayırmayı hızlandırmak için ek işaretçiler, ek bütünlük denetimi sağlamak için sihirli bir sayı ve diğer bilgileri içerebilir. Bölgenin boyutunu ve bunun gibi bir sihirli sayı içeren basit bir başlık varsayalım:

```
typedef struct {
    int size;
    int magic;
} header_t;
```

Yukarıdaki örnek, Şekil 17.2'de gördüğünüz gibi görünecektir. Kullanıcı `free(ptr)` öğesini çağırdığında, kitaplık başlığın nerede başladığını bulmak için basit işaretçi aritmetiğini kullanır.

```
void free(void *ptr) {
    header_t *hptr = (header_t *) ptr - 1;
    ...
```

Başlığa yönelik böyle bir işaretçi elde ettikten sonra, kitaplık, sihirli sayının tutarlılık kontrolü (`assert(hptr->magic == 1234567)`) ile beklenen değerle eşleşip eşleşmediğini kolayca belirleyebilir ve yeni serbest bırakılan bölgenin toplam boyutunu şu şekilde hesaplayabilir: basit matematik (yani, başlığın boyutunu bölgenin boyutuna eklemek). Son cümledeki küçük ama kritik ayrıntıya dikkat edin: serbest bölgenin boyutu, başlığın boyutu artı kullanıcıya ayrılan alanın boyutudur. Bu nedenle, bir kullanıcı N bayt bellek istediğinde, kitaplık N boyutunda boş bir öbek aramaz; bunun yerine, N boyutunda ve başlığın boyutunda boş bir yığın arar.

Boş Liste Gömme

Şimdiye kadar basit serbest listemizi kavramsal bir varlık olarak ele aldık; yığındaki boş bellek yığınlarını açıklayan bir listedir. Ancak boş alanın içinde böyle bir listeyi nasıl oluştururuz?

Daha tipik bir listede, yeni bir düğüm tahsis ederken, düğüm için alana ihtiyacınız olduğunda `malloc()` işlevini çağırırsınız. Ne yazık ki, bellek ayırma kitaplığında bunu yapamazsınız! Bunun yerine, listeyi boş alanın içinde oluşturmanız gerekir. Bu biraz garip geliyorsa endişelenmeyin; öyle, fakat yapamayacak kadar garip değil!

Yönetilecek 4096 baytlık bir bellek yığınımız olduğunu varsayalım (yani, yığın 4KB'dir). Bunu boş bir liste olarak yönetmek için öncelikle söz konusu listeyi başlatmalıyız; başlangıçta, listenin 4096 boyutunda (eksi başlık boyutu) bir girişi olmalıdır. İşte listedeki bir düğümün açıklaması:

```
typedef struct __node_t {
    int               size;
    struct __node_t *next;
} node_t;
```

Şimdi yığını başlatan ve boş listenin ilk elemanını bu boşluğa koyan bazı kodlara bakalım. Yığının, sistem çağrısı `mmap;`'e yapılan bir çağrı yoluyla elde edilen bir miktar boş alan içinde inşa edildiğini varsayıyoruz, bu tür bir yığın oluşturmanın tek yolu bu değildir, mesela bu örnekte bize yardım eder. İşte kod:

```
// mmap() returns a pointer to a chunk of free space
node_t *head = mmap(NULL, 4096, PROT_READ|PROT_WRITE,
                    MAP_ANON|MAP_PRIVATE, -1, 0);
head->size   = 4096 - sizeof(node_t);
head->next   = NULL;
```

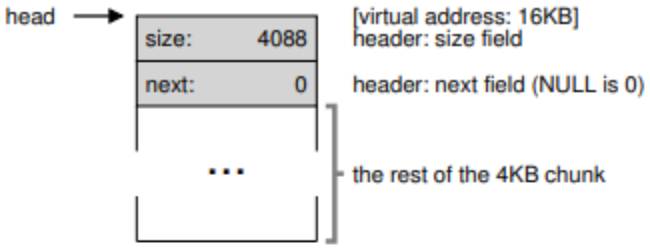


Figure 17.3: **A Heap With One Free Chunk**

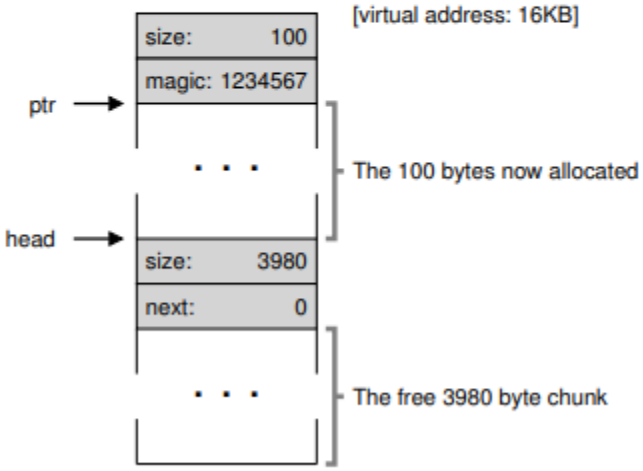


Figure 17.4: **A Heap: After One Allocation**

Bu kodu çalıştırdıktan sonra listenin durumu, 4088 boyutunda tek bir girdiye sahip olmasıdır. Evet, bu küçük bir yığın ama burada bizim için güzel bir örnek teşkil ediyor. Baş işaretçi, bu aralığın başlangıç adresini içerir; 16 KB olduğunu varsayalım (herhangi bir sanal adres iyi olsa da). Böylece yığın görsel olarak Şekil 17.3'te gördüğünüz gibi görünür.

Şimdi, diyelim ki 100 bayt boyutunda bir bellek yığınının istendiğini düşünelim. Bu isteğe hizmet vermek için, kitaplık önce isteği karşılayacak kadar büyük bir öbek bulacaktır; yalnızca bir boş parça olduğundan (boyut: 4088), bu yığın seçilecektir. Daha sonra yığın ikiye **bölünecektir(split)**: bir yığın, isteğe hizmet edecek kadar büyük (ve başlık yukarıda açılandığı gibi) ve kalan boş yığın. 8 baytlık bir başlığı (bir tamsayı boyutu ve bir tamsayı sihirli sayı) varsayarsak, yığındaki boşluk artık Şekil 17.4'te gördüğünüz gibi görünür.

Böylece,100 bayt talep edildiğinde,kütüphane 108 bayt tahsis etmiştir.

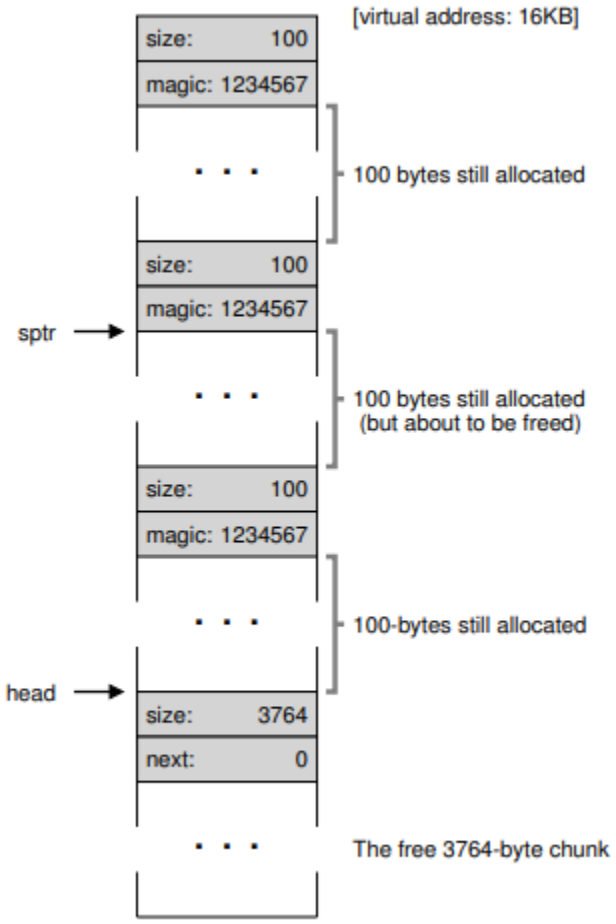


Figure 17.5: **Free Space With Three Chunks Allocated**

mevcut bir boş parçadan ona bir işaretçi (yukarıdaki şekilde `ptr` olarak işaretlenmiştir) döndürür, başlık bilgisini daha sonra `free()` sırasında kullanılmak üzere tahsis edilen alanın hemen önüne saklar ve listedeki bir boş düğümü 3980'e küçültür bayt (4088 eksi 108).

Şimdi, her biri 100 baytlık (veya başlık dahil 108) üç tahsis edilmiş bölge olduğunda yığına bakalım. Bu yığının bir görselleştirmesi Şekil 17.5'te gösterilmiştir.

Burada görebileceğiniz gibi, yığının ilk 324 baytı artık tahsis edilmiştir ve bu nedenle, bu alanda üç başlık ve ayrıca çağıran program tarafından kullanılan üç adet 100 baytlık bölge görüyoruz. boş liste ilginç olmaya devam ediyor: yalnızca tek bir düğüm (`başın` gösterdiği), ancak şimdi yalnızca 3764.

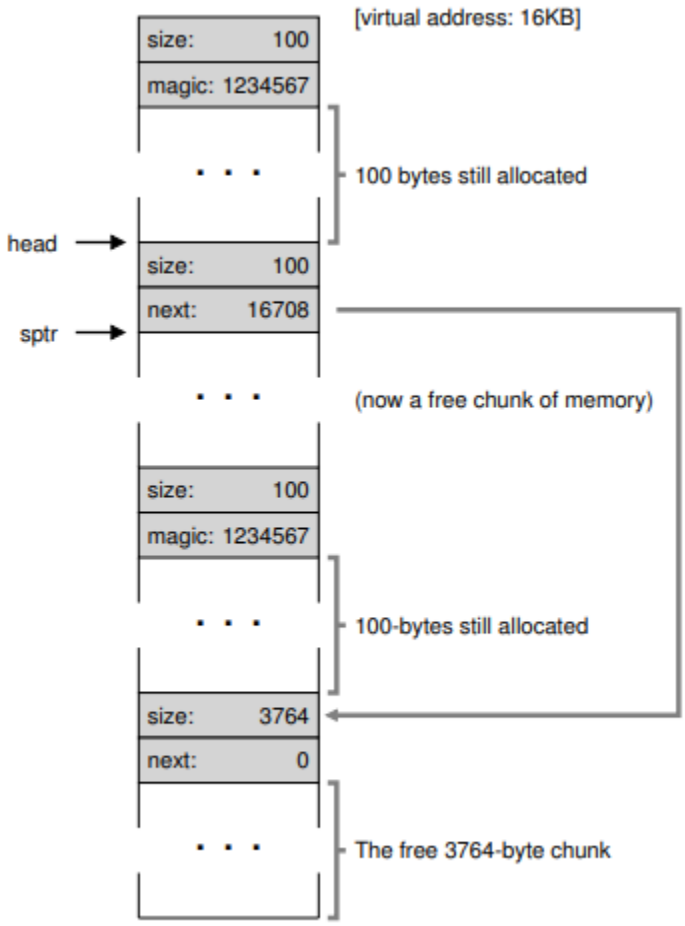


Figure 17.6: **Free Space With Two Chunks Allocated**

üç bölmeden sonra bayt boyutunda. Ancak, çağıran program `free()` aracılığıyla bir miktar bellek döndürdüğünde ne olur?

Bu örnekte uygulama, `free(16500)` öğesini çağırarak ayrılan belleğin orta yığınını döndürür (16500 değerine, bellek bölgesinin başlangıcı olan 16384, önceki yığının 108'ine ve 8 baytına eklenerek ulaşılır. bu parçanın başlığı). Bu değer önceki diyagramda `sptr` işaretçisi tarafından gösterilmiştir.

Kitaplık hemen serbest bölgenin boyutunu hesaplar ve ardından serbest öbeği tekrar serbest listeye ekler. Boş listenin başına yerleştirdiğimizi varsayarsak, boşluk şuna benzer (Şekil 17.6)

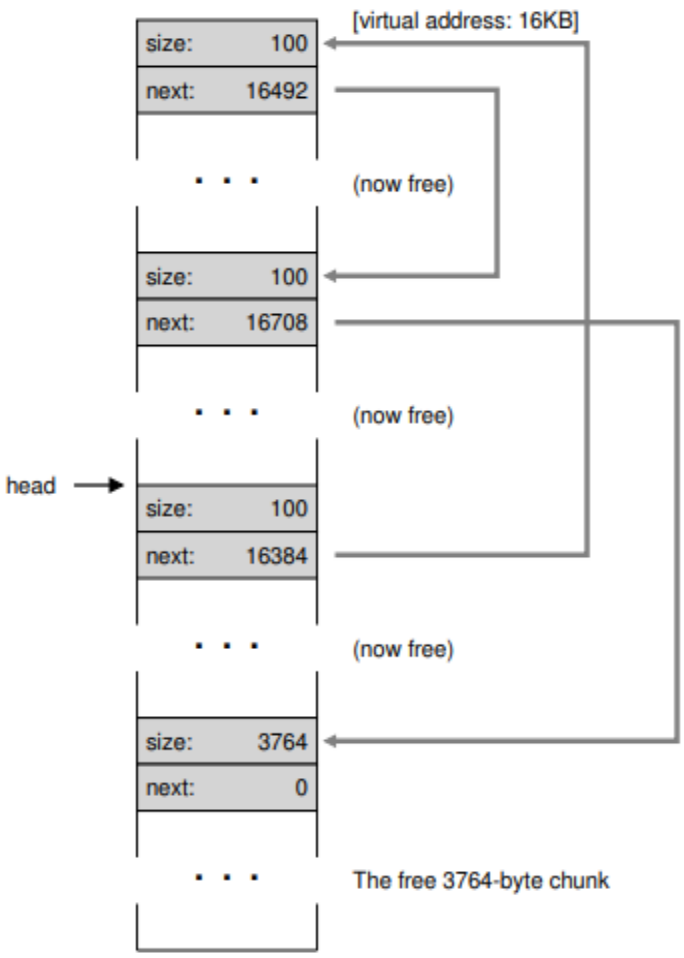


Figure 17.7: **A Non-Coalesced Free List**

Şimdi küçük bir boş öbek (listenin başında gösterilen 100 bayt) ve büyük bir boş yığın (3764 bayt) ile başlayan bir listemiz var. Sonunda listemizde birden fazla öğe var! Ve evet, boş alan parçalanmış, talihsiz ama yaygın bir olay

Son bir örnek: Şimdi kullanımda olan son iki parçanın serbest kaldığını varsayalım. Birleşme olmazsa parçalanma olur (Şekil 17.7).

Şekilden de görebileceğiniz gibi, şimdi büyük bir karmaşa yaşıyoruz! Neden? Niye? Basit, listeyi **birleştirmeyi(coalesce)** unuttuk. Hafızanın tamamı boş olmasına rağmen, parçalara ayrılmıştır, böylece bir olmamasına rağmen parçalanmış bir hafıza gibi görünür. Çözüm basit: listeyi gözden geçirin ve komşu parçaları **birleştirin(merge)**; bittiğinde, yığın tekrar bütün olacaktır.

Yığını Büyütmek

Pek çok ayırma kitaplığında bulunan son bir mekanizmayı ele almalıyız. Spesifik olarak, yığında yer kalmazsa ne yapmalısınız? En basit yaklaşım sadece başarısız olmaktır. Bazı durumlarda bu tek seçenektir ve bu nedenle NULL değerini döndürmek onurlu bir yaklaşımdır. Kendini kötü hissetme! Denedin ve başarısız olmana rağmen iyi mücadele ettin.

Geleneksel ayırıcıların çoğu küçük boyutlu bir yığınla başlar ve bittiğinde işletim sisteminden daha fazla bellek ister. Tipik olarak, bu, yığını büyütmek için bir tür sistem çağrısı (örneğin, çoğu UNIX sisteminde `sbrk`) yaptıkları ve ardından yeni parçaları oradan tahsis ettikleri anlamına gelir. İşletim sistemi, sbrk isteğine hizmet vermek için boş fiziksel sayfaları bulur, bunları istekte bulunan işlemin adres alanına eşler ve ardından yeni yığının sonunun değerini döndürür; bu noktada daha büyük bir yığın kullanılabilir ve istek başarıyla yerine getirilebilir.

17.3 Temel Stratejiler

Artık elimizde bazı makineler olduğuna göre, boş alanı yönetmek için bazı temel stratejilerin üzerinden geçelim. Bu yaklaşımlar çoğunlukla kendi başınıza düşünebileceğiniz oldukça basit ilkelere dayanmaktadır; Okumadan önce deneyin ve tüm alternatifleri (veya yenilerini)bulup bulamayacağınıza bakın.

İdeal ayırıcı hem hızlıdır hem de parçalanmayı en aza indirir. Ne yazık ki, tahsis akışı ve boş alan istekleri gelişigüzel olabileceğinden (sonuçta bunlar programcı tarafından belirlenir), herhangi bir belirli strateji, yanlış girdi seti verildiğinde oldukça kötü sonuç verebilir. Bu nedenle, “en iyi” yaklaşımı tarif etmeyeceğiz, bunun yerine bazı temel hususlardan bahsedeceğiz ve bunların artılarını ve eksilerini tartışacağız.

Best Fit

En uygun yaklaşım (best fit) oldukça basittir: ilk olarak, boş listede arama yapın ve istenen boyuttan daha büyük veya daha büyük olan boş bellek parçalarını bulun. Ardından, o aday grubundan en küçük olanı döndürün; bu sözde en uygun parçadır (en küçük uyum olarak da adlandırılabilir). Boş listeden bir kez geçmek, döndürülecek doğru bloğu bulmak için yeterlidir.

En uygun stratejinin ardındaki sezgi basittir: kullanıcının istediğine yakın bir blok döndürerek uygun strateji boşa harcanan alanı azaltmaya çalışır. Ancak bunun bir bedeli var; saf uygulamalar, doğru boş blok için kapsamlı bir arama gerçekleştirirken ağır bir performans cezası öder

Worst Fit

En kötü uyum(worst fit) yaklaşımı, en uygun yaklaşımın tersidir; en büyük parçayı bulun ve istenen miktarı iade edin; kalan (büyük) parçayı boş listede tutun. En kötü uyum, böylece en uygun yaklaşımdan doğabilecek çok sayıda ki dalların yerine,

büyük parçaları serbest bırakmaya çalışır. Ancak bir kez daha, tam bir boş alan araması gereklidir ve bu nedenle bu yaklaşım maliyetli olabilir. Daha da kötüsü, çoğu araştırma, kötü performans gösterdiğini, bunun da yüksek genel giderlere sahipken aşırı parçalanmaya yol açtığını gösteriyor.

First fit

İlk sığdırma(first fit) yöntemi, yeterince büyük olan ilk bloğu bulur ve istenen miktarı kullanıcıya iade eder. Daha önce olduğu gibi, kalan boş alan sonraki istekler için boş tutulur.

İlk sığdırmanın hız avantajı vardır - tüm boş alanların kapsamlı bir şekilde aranması gerekmez - ancak bazen boş listenin başlangıcını küçük nesnelerle kirletir. Böylece, ayırıcının serbest liste sırasını nasıl yönettiği bir sorun haline gelir. Bir yaklaşım, **adrese dayalı sıralamayı(address-based ordering)** kullanmaktır; listeyi boş alanın adresine göre sıralayarak, birleştirme daha kolay hale gelir ve parçalanma azalma eğilimi gösterir.

Next fit

İlk sığdırma yöntemi gibi her zaman listenin başında başlatmak yerine, **sonraki sığdırma(next fit)** yönteminin algoritmas:Listede en son bakılan konuma fazladan bir işaretçi tutmak;. boş alan aramalarını liste boyunca daha düzgün bir şekilde yaymak ve böylece listenin başındaki parçalanmayı önlemektir. Böyle bir yaklaşımın performansı, Kapsamlı bir aramadan bir kez daha kaçınıldığı için, ilk sığdırma yöntemine oldukça benzer.

Örnekler

İşte yukarıdaki stratejilere birkaç örnek. 10, 30 ve 20 boyutlarında üç öğe içeren ücretsiz bir liste tasarlayın edin (burada sadece stratejilerin nasıl çalıştığına odaklanmak yerine başlıkları ve diğer ayrıntıları göz ardı edeceğiz):

15 büyüklüğünde bir tahsis talebi varsayalım. En uygun yaklaşım, tüm listeyi arayın ve talebi karşılayabilecek en küçük boş alan olduğundan 20'nin en uygun olduğunu bulun. Ortaya çıkan boş liste:

Bu örnekte olduğu gibi ve çoğu zaman en uygun yaklaşımda olduğu gibi, artık küçük bir boş yığın kalmıştır. En kötü uyum yaklaşımı benzerdir ancak bunun yerine bu örnekte 30 olan en büyük parçayı bulur. Ortaya çıkan liste:

Bu örnekteki ilk uygun strateji, en kötü uyum stratejisiyle aynı şeyi yapar ve ayrıca talebi karşılayabilecek ilk serbest bloğu bulur. Fark, arama maliyetindedir; hem en uygun hem de en kötü uyum tüm listeye bakar; ilk uygun strateji uygun olanı bulana kadar yalnızca boş parçaları inceler, böylece arama maliyetini düşürür.

Bu örnekler, tahsisat politikalarının yüzeyini çiziyor. Daha derin bir anlayış için gerçek iş yükleri ve daha karmaşık ayırıcı davranışları (örneğin birleştirme) ile daha ayrıntılı analizler gereklidir. Belki bir ev ödevi bölümü için bir şey diyorsundur?

17.4 Diğer yaklaşımlar

Yukarıda açıklanan temel yaklaşımların ötesinde, bellek tahsisini bir şekilde iyileştirmek için önerilen bir dizi teknik ve algoritma vardır. Göz önünde bulundurmanız için birkaçını burada listeliyoruz (yani, en uygun tahsisten biraz daha fazlasını düşünmenizi sağlamak için)

Ayrılmış Listeler

Bir süredir ortalıkta dolaşan ilginç bir yaklaşım, **ayrılmış listelerin(segregated lists)** kullanılmasıdır. Temel fikir basittir: Belirli bir uygulamanın yaptığı popüler boyutta bir (veya birkaç) istek varsa, yalnızca o boyuttaki nesneleri yönetmek için ayrı bir liste tutun; diğer tüm istekler daha genel bir bellek ayırıcıya iletilir.

Böyle bir yaklaşımın yararları açıktır. Belirli bir boyuttaki istekler için ayrılmış bir bellek parçasına sahip olarak, parçalanma çok daha az endişe vericidir; ayrıca, karmaşık bir liste araması gerekmediğinden, tahsis ve boş alan istekleri doğru boyutta olduklarında oldukça hızlı bir şekilde sunulabilir.

Her iyi fikir gibi, bu yaklaşım da bir sisteme yeni komplikasyonlar getirir. Örneğin, genel havuzun aksine, belirli bir boyuttaki özel isteklere hizmet eden bellek havuzuna ne kadar bellek ayrılmalıdır? Belirli bir ayırıcı, uber mühendisi Jeff Bonwick'in (Solaris çekirdeğinde kullanılmak üzere tasarlanmış) **döşeme ayırıcısı(slab allocator)**, bu sorunu oldukça güzel bir şekilde ele alıyor [B94].

Spesifik olarak, çekirdek önyüklendiğinde, sık sık talep edilmesi muhtemel çekirdek nesneleri (kilitler, dosya sistemi düğümleri, vb.) için bir dizi **nesne önbelleği(object caches)** tahsis eder; bu nedenle nesne önbelleklerinin her biri, belirli bir boyutta ayrılmış boş listelerdir ve hızlı bir şekilde bellek tahsisi ve serbest istekler sunar. Belirli bir önbellekte boş alan azaldığında, daha genel bir bellek ayırıcıdan bazı **bellek dilimleri(slabs)** ister (istenen toplam miktar, sayfa boyutunun ve söz konusu nesnenin katıdır). Tersine, belirli bir levha içindeki nesnelerin referans sayıları sıfıra gittiğinde, genel ayırıcı bunları uzmanlaşmış ayırıcıdan geri alabilir, bu genellikle VM sistemi daha fazla belleğe ihtiyaç duyduğunda yapılır.

Apar:Harika Mühendisler Gerçekten Harika

Jeff Bonwick gibi mühendisler (yalnızca burada bahsedilen döşeme ayırıcıyı yazmakla kalmayıp aynı zamanda harika bir dosya sistemi olan ZFS'nin de lideriydi) Silikon Vadisi'nin kalbidir. Hemen hemen her harika ürünün veya teknolojinin arkasında, yetenekleri, yetenekleri ve bağlılıkları açısından ortalamanın çok üzerinde olan bir insan (veya küçük bir insan grubu) vardır. Mark Zuckerberg'in (Facebook'tan) dediği gibi: "Rolünde istisnai olan biri, oldukça iyi olan birinden sadece biraz daha iyi değildir. 100 kat daha iyidir" Bu nedenle, bugün hala bir veya iki kişi dünyanın çehresini sonsuza dek değiştiren bir şirket kurabilir (Google, Apple veya Facebook'u düşünün). Çok çalışın ve siz de böyle bir "100x" kişi olabilirsiniz. Aksi takdirde, böyle bir kişiyle çalışın; çoğu kişinin bir ayda öğrendiğinden daha fazlasını bir günde öğreneceksiniz. Bunu başaramazsan, üzgün hissedin.

Döşeme ayırıcı ayrıca, listelerdeki boş nesneleri önceden başlatılmış bir durumda tutarak ayrılmış liste yaklaşımlarının çoğunun ötesine geçer. Bonwick, veri yapılarının başlatılmasının ve yok edilmesinin maliyetli olduğunu gösteriyor [B94]; Serbest bırakılan nesneleri belirli bir listede başlatılmış durumlarında tutarak, döşeme ayırıcı böylece nesne başına sık başlatma ve yok etme döngülerini önler ve böylece genel giderleri önemli ölçüde azaltır.

Dost Tahsisi(Ayırıcı)

Birleştirme bir ayırıcı için kritik olduğundan, birleştirme işlemini basit hale getirmek için bazı yaklaşımlar tasarlanmıştır. **İkili arkadaş ayırıcıda(binary buddy allocator**) [K65] iyi bir örnek bulunur.

Böyle bir sistemde, boş bellek önce kavramsal olarak 2 N büyüklüğünde büyük bir alan olarak düşünülür. Bellek için bir talep yapıldığında, boş alan araması, talebi karşılamak için yeterince büyük bir blok bulunana kadar boş alanı yinelemeli olarak ikiye böler (ve ikiye bölmek, çok küçük bir alana neden olur). Bu noktada istenen blok kullanıcıya geri döner. Burada, 7 KB'lık bir blok aranırken bölünen 64 KB'lik boş alanın bir örneği verilmiştir (Şekil 17.8, sayfa 15)

Örnekte, en soldaki 8 KB'lik blok tahsis edilmiş (grinin daha koyu tonuyla gösterildiği gibi) ve kullanıcıya geri dönmüştür; yalnızca iki boyutlu blokların gücünü vermenize izin verildiğinden, bu şemanın **dahili parçalanmadan(internal fragmentation)** zarar görebileceğini unutmayın.

Arkadaş tahsisinin güzelliği, o blok serbest bırakıldığında olanlarda bulunur. 8KB bloğunu boş listeye geri döndürürken, ayırıcı "dost" 8KB'nin boş olup olmadığını kontrol eder; öyleyse, iki bloğu 16 KB'lik bir blokta birleştirir. Ayırıcı daha sonra 16KB bloğunun arkadaşının hala boş olup olmadığını kontrol eder; eğer öyleyse, bu iki bloğu birleştirir. Bu yinelemeli birleştirme işlemi, ya tüm boş alanı geri yükleyerek ya da bir arkadaşın kullanımda olduğu tespit edildiğinde durarak parçada devam eder.

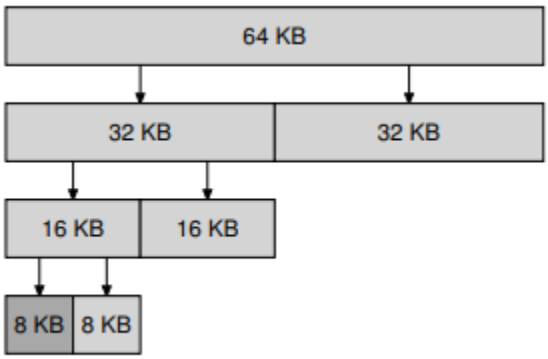


Figure 17.8: **Example Buddy-managed Heap**

Arkadaş tahsisinin bu kadar iyi çalışmasının nedeni, belirli bir bloğun arkadaşını belirlemenin basit olmasıdır. Nasıl soruyorsun? Yukarıdaki boş alanda bulunan blokların adreslerini düşününüz. Yeterince dikkatli düşünürseniz, her bir arkadaş çiftinin adresinin yalnızca bir bit farklı olduğunu görürsünüz; hangi bit, dostum ağacındaki seviyeye göre belirlenir. Böylece, ikili arkadaş tahsis planlarının nasıl çalıştığına dair temel bir fikriniz olur. Daha fazla ayrıntı için, her zaman olduğu gibi, Wilson anketine bakın [W+95]

Diğerler fikirler

Yukarıda açıklanan yaklaşımların çoğuyla ilgili önemli bir sorun, **ölçeklendirme(scaling)** eksikliğidir. Özellikle, arama listeleri oldukça yavaş olabilir. Bu nedenle, gelişmiş ayırıcılar, bu maliyetleri ele almak için daha karmaşık veri yapıları kullanır ve basitliği performansla değiştirir. Örnekler arasında dengeli ikili ağaçlar, yayvan ağaçlar veya kısmen sıralı ağaçlar bulunur [W+95].

Modern sistemlerin genellikle birden çok işlemciye sahip olduğu ve çok iş parçacıklı iş yükleri çalıştırdığı göz önüne alındığında (kitabın Eşzamanlılık bölümünde ayrıntılı olarak öğreneceğiniz bir şey), ayırıcıların iyi çalışması için çok fazla çaba harcanması şaşırtıcı değildir. çok işlemcili sistemlerde Berger ve diğerlerinde iki harika örnek bulunur. [B+00] ve Evans [E06]; ayrıntılar için onları kontrol edin.

Bunlar, insanların bellek ayırıcılar hakkında zaman içinde sahip oldukları binlerce fikirden yalnızca ikisidir; merak ediyorsan kendi kendine oku. Bunu başaramazsanız, size gerçek dünyanın nasıl bir yer olduğuna dair bir fikir vermesi için glibc ayırıcının nasıl çalıştığını [S15] okuyun.

17.15 Özet

Bu bölümde, bellek ayırıcıların en basit biçimlerini tartıştık. Bu tür ayırıcılar her yerde bulunur, yazdığınız her C programına ve ayrıca kendi veri yapıları için belleği yöneten işletim sistemine bağlıdır.

Pek çok sistemde olduğu gibi, böyle bir sistemi oluştururken yapılacak birçok değiş tokuş vardır ve bir ayırıcıya sunulan tam iş yükü hakkında ne kadar çok şey bilirseniz, onu o iş yükü için daha iyi çalışacak şekilde ayarlamak için o kadar çok şey yapabilirsiniz. Çok çeşitli iş yükleri için iyi çalışan, hızlı, alan açısından verimli, ölçeklenebilir bir ayırıcı yapmak, modern bilgisayar sistemlerinde süregelen bir zorluk olmaya devam ediyor.

References

[B+00] "Hoard: A Scalable Memory Allocator for Multithreaded Applications" by Emery D. Berger, Kathryn S. McKinley, Robert D. Blumofe, Paul R. Wilson. ASPLOS-IX, November 2000. Berger and company's excellent allocator for multiprocessor systems. Beyond just being a fun paper, also used in practice!

[B94] "The Slab Allocator: An Object-Caching Kernel Memory Allocator" by Jeff Bonwick. USENIX '94. A cool paper about how to build an allocator for an operating system kernel, and a great example of how to specialize for particular common object sizes.

[E06] "A Scalable Concurrent malloc(3) Implementation for FreeBSD" by Jason Evans. April, 2006. http://people.freebsd.org/~jasone/jemalloc/bsdcan2006/jemalloc.pdf. A detailed look at how to build a real modern allocator for use in multiprocessors. The "jemalloc" allocator is in widespread use today, within FreeBSD, NetBSD, Mozilla Firefox, and within Facebook.

[K65] "A Fast Storage Allocator" by Kenneth C. Knowlton. Communications of the ACM, Volume 8:10, October 1965. The common reference for buddy allocation. Random strange fact: Knuth gives credit for the idea not to Knowlton but to Harry Markowitz, a Nobel-prize winning economist. Another strange fact: Knuth communicates all of his emails via a secretary; he doesn't send email himself, rather he tells his secretary what email to send and then the secretary does the work of emailing. Last Knuth fact: he created TeX, the tool used to typeset this book. It is an amazing piece of software4

[S15] "Understanding glibc malloc" by Sploitfun. February, 2015. sploitfun.wordpress.com/2015/02/10/understanding-glibc-malloc/. A deep dive into how glibc malloc works. Amazingly detailed and a very cool read

[W+95] "Dynamic Storage Allocation: A Survey and Critical Review" by Paul R. Wilson, Mark S. Johnstone, Michael Neely, David Boles. International Workshop on Memory Management, Scotland, UK, September 1995. An excellent and far-reaching survey of many facets of memory allocation. Far too much detail to go into in this tiny chapter!

ÖDEV(SİMÜLASYON)

1- Birkaç rasgele ayırma ve serbest bırakma oluşturmak için önce `-n 10 -H 0 p BEST -s 0` bayraklarıyla çalıştırın. alloc()/free() öğesinin ne döndüreceğini tahmin edebilir misiniz? Her istekten sonra boş listenin durumunu tahmin edebilir misiniz Zaman içinde ücretsiz liste hakkında ne fark ettiniz?

> Fiziksel bellek küçük parçalara bölünür.Sonunda birleşme olmadığı için de "1" büyüklüğünde boş alan elde ederiz.

```
onur@onur:~/Desktop$ python3 ./malloc.py -n 10 -H 0 -p BEST -s 0 -c
seed 0
size 100
baseAddr 1000
headerSize 0
alignment -1
policy BEST
listOrder ADDRSORT
coalesce False
numOps 10
range 10
percentAlloc 50
allocList
compute True

ptr[0] = Alloc(3) returned 1000 (searched 1 elements)
Free List [ Size 1 ]: [ addr:1003 sz:97 ]

Free(ptr[0])
returned 0
Free List [ Size 2 ]: [ addr:1000 sz:3 ][ addr:1003 sz:97 ]

ptr[1] = Alloc(5) returned 1003 (searched 2 elements)
Free List [ Size 2 ]: [ addr:1000 sz:3 ][ addr:1008 sz:92 ]

Free(ptr[1])
returned 0
Free List [ Size 3 ]: [ addr:1000 sz:3 ][ addr:1003 sz:5 ][ addr:1008 sz:92 ]

ptr[2] = Alloc(8) returned 1008 (searched 3 elements)
Free List [ Size 3 ]: [ addr:1000 sz:3 ][ addr:1003 sz:5 ][ addr:1016 sz:84 ]

Free(ptr[2])
returned 0
Free List [ Size 4 ]: [ addr:1000 sz:3 ][ addr:1003 sz:5 ][ addr:1008 sz:8 ][ addr:1016 sz:84 ]

ptr[3] = Alloc(8) returned 1008 (searched 4 elements)
Free List [ Size 3 ]: [ addr:1000 sz:3 ][ addr:1003 sz:5 ][ addr:1016 sz:84 ]

Free(ptr[3])
returned 0
Free List [ Size 4 ]: [ addr:1000 sz:3 ][ addr:1003 sz:5 ][ addr:1008 sz:8 ][ addr:1016 sz:84 ]

ptr[4] = Alloc(2) returned 1000 (searched 4 elements)
Free List [ Size 4 ]: [ addr:1002 sz:1 ][ addr:1003 sz:5 ][ addr:1008 sz:8 ][ addr:1016 sz:84 ]

ptr[5] = Alloc(7) returned 1008 (searched 4 elements)
Free List [ Size 4 ]: [ addr:1002 sz:1 ][ addr:1003 sz:5 ][ addr:1015 sz:1 ][ addr:1016 sz:84 ]
```

2- Boş listede arama yapmak için EN KÖTÜ sığdırma ilkesi kullanıldığında sonuçlar nasıl farklıdır (`-p BEST`) Ne değişir?

> Bellek daha fazla parçaya ayrılır ve daha fazla öğe aranır.

```
onur@onur:~$ python3 ./malloc.py -n 10 -H 0 -p WORST -s 0 -c
seed 0
size 100
baseAddr 1000
headerSize 0
alignment -1
policy WORST
listOrder ADDRSORT
coalesce False
numOps 10
range 10
percentAlloc 50
allocList
compute True

ptr[0] = Alloc(3) returned 1000 (searched 1 elements)
Free List [ Size 1 ]: [ addr:1003 sz:97 ]

Free(ptr[0])
returned 0
Free List [ Size 2 ]: [ addr:1000 sz:3 ][ addr:1003 sz:97 ]

ptr[1] = Alloc(5) returned 1003 (searched 2 elements)
Free List [ Size 2 ]: [ addr:1000 sz:3 ][ addr:1008 sz:92 ]

Free(ptr[1])
returned 0
Free List [ Size 3 ]: [ addr:1000 sz:3 ][ addr:1003 sz:5 ][ addr:1008 sz:92 ]

ptr[2] = Alloc(8) returned 1008 (searched 3 elements)
Free List [ Size 3 ]: [ addr:1000 sz:3 ][ addr:1003 sz:5 ][ addr:1016 sz:84 ]

Free(ptr[2])
returned 0
Free List [ Size 4 ]: [ addr:1000 sz:3 ][ addr:1003 sz:5 ][ addr:1008 sz:8 ][ addr:1016 sz:84 ]

ptr[3] = Alloc(8) returned 1016 (searched 4 elements)
Free List [ Size 4 ]: [ addr:1000 sz:3 ][ addr:1003 sz:5 ][ addr:1008 sz:8 ][ addr:1024 sz:76 ]

Free(ptr[3])
returned 0
Free List [ Size 5 ]: [ addr:1000 sz:3 ][ addr:1003 sz:5 ][ addr:1008 sz:8 ][ addr:1016 sz:8 ][ addr:1024 sz:76 ]

ptr[4] = Alloc(2) returned 1024 (searched 5 elements)
Free List [ Size 5 ]: [ addr:1000 sz:3 ][ addr:1003 sz:5 ][ addr:1008 sz:8 ][ addr:1016 sz:8 ][ addr:1026 sz:74 ]

ptr[5] = Alloc(7) returned 1026 (searched 5 elements)
Free List [ Size 5 ]: [ addr:1000 sz:3 ][ addr:1003 sz:5 ][ addr:1008 sz:8 ][ addr:1016 sz:8 ][ addr:1033 sz:67 ]
```

3-FIRST fit (-p FIRST) kullanılırken ne olur? First fit'i kullandığınızda ne hızlanır

>En iyi veya en kötü eşleşmeyi aramaya gerek duyulmaz bu yüzden aramayı ve Tahsisi hızlandırır.

```
onur@onur:~/Desktop$ python3 ./malloc.py -n 10 -H 0 -p FIRST -s 0 -c
seed 0
size 100
baseAddr 1000
headerSize 0
alignment -1
policy FIRST
listOrder ADDRSORT
coalesce False
numOps 10
range 10
percentAlloc 50
allocList
compute True

ptr[0] = Alloc(3) returned 1000 (searched 1 elements)
Free List [ Size 1 ]: [ addr:1003 sz:97 ]

Free(ptr[0])
returned 0
Free List [ Size 2 ]: [ addr:1000 sz:3 ][ addr:1003 sz:97 ]

ptr[1] = Alloc(5) returned 1003 (searched 2 elements)
Free List [ Size 2 ]: [ addr:1000 sz:3 ][ addr:1008 sz:92 ]

Free(ptr[1])
returned 0
Free List [ Size 3 ]: [ addr:1000 sz:3 ][ addr:1003 sz:5 ][ addr:1008 sz:92 ]

ptr[2] = Alloc(8) returned 1008 (searched 3 elements)
Free List [ Size 3 ]: [ addr:1000 sz:3 ][ addr:1003 sz:5 ][ addr:1016 sz:84 ]

Free(ptr[2])
returned 0
Free List [ Size 4 ]: [ addr:1000 sz:3 ][ addr:1003 sz:5 ][ addr:1008 sz:8 ][ addr:1016 sz:84 ]

ptr[3] = Alloc(8) returned 1008 (searched 3 elements)
Free List [ Size 3 ]: [ addr:1000 sz:3 ][ addr:1003 sz:5 ][ addr:1016 sz:84 ]

Free(ptr[3])
returned 0
Free List [ Size 4 ]: [ addr:1000 sz:3 ][ addr:1003 sz:5 ][ addr:1008 sz:8 ][ addr:1016 sz:84 ]

ptr[4] = Alloc(2) returned 1000 (searched 1 elements)
Free List [ Size 4 ]: [ addr:1002 sz:1 ][ addr:1003 sz:5 ][ addr:1008 sz:8 ][ addr:1016 sz:84 ]

ptr[5] = Alloc(7) returned 1008 (searched 3 elements)
Free List [ Size 4 ]: [ addr:1002 sz:1 ][ addr:1003 sz:5 ][ addr:1015 sz:1 ][ addr:1016 sz:84 ]
```

4- Yukarıdaki sorular için, listenin nasıl düzenli tutulduğu bazı ilkeler için boş yer bulma süresini etkileyebilir. İlkelerin ve liste sıralamalarının nasıl etkileşime girdiğini görmek için farklı ücretsiz liste sıralamalarını (-l ADDRSORT, -l SIZESORT+, -l SIZESORT-) kullanın.

>İlk uyum, listenin sıralama şeklinden etkilenmez. En iyi ve en kötü uyum sırasıyla SIZESORT+ ve SIZESORT-'tan yararlanır. Ancak birleşmemiz olsaydı, ADRSORT birleşmeye yardımcı olurdu.

```
onur@onur:~/Desktop$ python3 ./malloc.py -p BEST -l SIZESORT+ -c
seed 0
size 100
baseAddr 1000
headerSize 0
alignment -1
policy BEST
listOrder SIZESORT+
coalesce False
numOps 10
range 10
percentAlloc 50
allocList
compute True

ptr[0] = Alloc(3) returned 1000 (searched 1 elements)
Free List [ Size 1 ]: [ addr:1003 sz:97 ]

Free(ptr[0])
returned 0
Free List [ Size 2 ]: [ addr:1000 sz:3 ][ addr:1003 sz:97 ]

ptr[1] = Alloc(5) returned 1003 (searched 2 elements)
Free List [ Size 2 ]: [ addr:1000 sz:3 ][ addr:1008 sz:92 ]

Free(ptr[1])
returned 0
Free List [ Size 3 ]: [ addr:1000 sz:3 ][ addr:1003 sz:5 ][ addr:1008 sz:92 ]

ptr[2] = Alloc(8) returned 1008 (searched 3 elements)
Free List [ Size 3 ]: [ addr:1000 sz:3 ][ addr:1003 sz:5 ][ addr:1016 sz:84 ]

Free(ptr[2])
returned 0
Free List [ Size 4 ]: [ addr:1000 sz:3 ][ addr:1003 sz:5 ][ addr:1008 sz:8 ][ addr:1016 sz:84 ]

ptr[3] = Alloc(8) returned 1008 (searched 4 elements)
Free List [ Size 3 ]: [ addr:1000 sz:3 ][ addr:1003 sz:5 ][ addr:1016 sz:84 ]

Free(ptr[3])
returned 0
Free List [ Size 4 ]: [ addr:1000 sz:3 ][ addr:1003 sz:5 ][ addr:1008 sz:8 ][ addr:1016 sz:84 ]

ptr[4] = Alloc(2) returned 1000 (searched 4 elements)
Free List [ Size 4 ]: [ addr:1002 sz:1 ][ addr:1003 sz:5 ][ addr:1008 sz:8 ][ addr:1016 sz:84 ]

ptr[5] = Alloc(7) returned 1008 (searched 4 elements)
Free List [ Size 4 ]: [ addr:1002 sz:1 ][ addr:1003 sz:5 ][ addr:1015 sz:1 ][ addr:1016 sz:84 ]
```

```
onur@onur:~/Desktop$ python3 ./malloc.py -p WORST -l SIZESORT- -c
seed 0
size 100
baseAddr 1000
headerSize 0
alignment -1
policy WORST
listOrder SIZESORT-
coalesce False
numOps 10
range 10
percentAlloc 50
allocList
compute True

ptr[0] = Alloc(3) returned 1000 (searched 1 elements)
Free List [ Size 1 ]: [ addr:1003 sz:97 ]

Free(ptr[0])
returned 0
Free List [ Size 2 ]: [ addr:1003 sz:97 ][ addr:1000 sz:3 ]

ptr[1] = Alloc(5) returned 1003 (searched 2 elements)
Free List [ Size 2 ]: [ addr:1008 sz:92 ][ addr:1000 sz:3 ]

Free(ptr[1])
returned 0
Free List [ Size 3 ]: [ addr:1008 sz:92 ][ addr:1003 sz:5 ][ addr:1000 sz:3 ]

ptr[2] = Alloc(8) returned 1008 (searched 3 elements)
Free List [ Size 3 ]: [ addr:1016 sz:84 ][ addr:1003 sz:5 ][ addr:1000 sz:3 ]

Free(ptr[2])
returned 0
Free List [ Size 4 ]: [ addr:1016 sz:84 ][ addr:1008 sz:8 ][ addr:1003 sz:5 ][ addr:1000 sz:3 ]

ptr[3] = Alloc(8) returned 1016 (searched 4 elements)
Free List [ Size 4 ]: [ addr:1024 sz:76 ][ addr:1008 sz:8 ][ addr:1003 sz:5 ][ addr:1000 sz:3 ]

Free(ptr[3])
returned 0
Free List [ Size 5 ]: [ addr:1024 sz:76 ][ addr:1008 sz:8 ][ addr:1016 sz:8 ][ addr:1003 sz:5 ][ addr:1000 sz:3 ]

ptr[4] = Alloc(2) returned 1024 (searched 5 elements)
Free List [ Size 5 ]: [ addr:1026 sz:74 ][ addr:1008 sz:8 ][ addr:1016 sz:8 ][ addr:1003 sz:5 ][ addr:1000 sz:3 ]

ptr[5] = Alloc(7) returned 1026 (searched 5 elements)
Free List [ Size 5 ]: [ addr:1033 sz:67 ][ addr:1008 sz:8 ][ addr:1016 sz:8 ][ addr:1003 sz:5 ][ addr:1000 sz:3 ]
```

```
onur@onur:~/Desktop$ python3 ./malloc.py -p FIRST -l SIZESORT+ -c
seed 0
size 100
baseAddr 1000
headerSize 0
alignment -1
policy FIRST
listOrder SIZESORT+
coalesce False
numOps 10
range 10
percentAlloc 50
allocList
compute True

ptr[0] = Alloc(3) returned 1000 (searched 1 elements)
Free List [ Size 1 ]: [ addr:1003 sz:97 ]

Free(ptr[0])
returned 0
Free List [ Size 2 ]: [ addr:1000 sz:3 ][ addr:1003 sz:97 ]

ptr[1] = Alloc(5) returned 1003 (searched 2 elements)
Free List [ Size 2 ]: [ addr:1000 sz:3 ][ addr:1008 sz:92 ]

Free(ptr[1])
returned 0
Free List [ Size 3 ]: [ addr:1000 sz:3 ][ addr:1003 sz:5 ][ addr:1008 sz:92 ]

ptr[2] = Alloc(8) returned 1008 (searched 3 elements)
Free List [ Size 3 ]: [ addr:1000 sz:3 ][ addr:1003 sz:5 ][ addr:1016 sz:84 ]

Free(ptr[2])
returned 0
Free List [ Size 4 ]: [ addr:1000 sz:3 ][ addr:1003 sz:5 ][ addr:1008 sz:8 ][ addr:1016 sz:84 ]

ptr[3] = Alloc(8) returned 1008 (searched 3 elements)
Free List [ Size 3 ]: [ addr:1000 sz:3 ][ addr:1003 sz:5 ][ addr:1016 sz:84 ]

Free(ptr[3])
returned 0
Free List [ Size 4 ]: [ addr:1000 sz:3 ][ addr:1003 sz:5 ][ addr:1008 sz:8 ][ addr:1016 sz:84 ]

ptr[4] = Alloc(2) returned 1000 (searched 1 elements)
Free List [ Size 4 ]: [ addr:1002 sz:1 ][ addr:1003 sz:5 ][ addr:1008 sz:8 ][ addr:1016 sz:84 ]

ptr[5] = Alloc(7) returned 1008 (searched 3 elements)
Free List [ Size 4 ]: [ addr:1002 sz:1 ][ addr:1003 sz:5 ][ addr:1015 sz:1 ][ addr:1016 sz:84 ]
```

5- Boş bir listenin birleştirilmesi oldukça önemli olabilir. Rastgele tahsis sayısını artırın (`-n 1000`'e söyleyin). Zamanla daha büyük ayırma isteklerine ne olur? Birleştirme ile ve olmadan çalıştırın (yani, `-C` bayrağıyla ve olmadan). Sonuçta ne gibi farklılıklar görüyorsunuz? Her durum için boş liste ne kadar büyük? Bu durumda listenin sıralaması önemli midir?

>Birleştirme olmadan ve daha yüksek -n ile 1 boyutlu boş alan elde ederiz,ki hiç kullanışlı değil.

Birleştirme açıkken, ADRSORT tüm uydurma ilkeleriyle en iyi sonucu verir.

***En kötü uyum, her iki sıralamada da kötü performans gösterir. Bunun nedeni, daha büyük boş alanların tamamı kesilene kadar daha küçük boş alanların kullanılmamasıdır. Zamanla sıralama ile bu boş alanlar geçici alanlarından ayrılır. Bu durum onların birleşmesine izin vermez. Farklı bir birleştirme algoritması bu sorunu çözebilir, ancak birleştirme çok daha fazla zaman alacaktır.**

***Best fit, her iki sıralamada da benzer şekilde çalışır. Çünkü daha küçük alanlar daha sık kullanılır ve bu nedenle bulaşıcı alanlar, en kötü uyum yaklaşımına göre daha fazla bir arada kalır.**

***Boyut sıralamasıyla ilk uyum, ya en kötü uyum ya da en iyi uyumdur. Yani, onların sonucuyla aynı sonucu verir.**

```
onur@onur:~/Desktop$ python3 ./malloc.py -n 1000 -r 30 -c
seed 0
size 100
baseAddr 1000
headerSize 0
alignment -1
policy BEST
listOrder ADDRSORT
coalesce False
numOps 1000
range 30
percentAlloc 50
allocList
compute True

ptr[0] = Alloc(8) returned 1000 (searched 1 elements)
Free List [ Size 1 ]: [ addr:1008 sz:92 ]

Free(ptr[0])
returned 0
Free List [ Size 2 ]: [ addr:1000 sz:8 ][ addr:1008 sz:92 ]

ptr[1] = Alloc(15) returned 1008 (searched 2 elements)
Free List [ Size 2 ]: [ addr:1000 sz:8 ][ addr:1023 sz:77 ]

Free(ptr[1])
returned 0
Free List [ Size 3 ]: [ addr:1000 sz:8 ][ addr:1008 sz:15 ][ addr:1023 sz:77 ]

ptr[2] = Alloc(23) returned 1023 (searched 3 elements)
Free List [ Size 3 ]: [ addr:1000 sz:8 ][ addr:1008 sz:15 ][ addr:1046 sz:54 ]

Free(ptr[2])
returned 0
Free List [ Size 4 ]: [ addr:1000 sz:8 ][ addr:1008 sz:15 ][ addr:1023 sz:23 ][ addr:1046 sz:54 ]

ptr[3] = Alloc(22) returned 1023 (searched 4 elements)
Free List [ Size 4 ]: [ addr:1000 sz:8 ][ addr:1008 sz:15 ][ addr:1045 sz:1 ][ addr:1046 sz:54 ]

Free(ptr[3])
returned 0
Free List [ Size 5 ]: [ addr:1000 sz:8 ][ addr:1008 sz:15 ][ addr:1023 sz:22 ][ addr:1045 sz:1 ][ addr:1046 sz:54 ]

ptr[4] = Alloc(4) returned 1000 (searched 5 elements)
Free List [ Size 5 ]: [ addr:1004 sz:4 ][ addr:1008 sz:15 ][ addr:1023 sz:22 ][ addr:1045 sz:1 ][ addr:1046 sz:54 ]

ptr[5] = Alloc(19) returned 1023 (searched 5 elements)
Free List [ Size 5 ]: [ addr:1004 sz:4 ][ addr:1008 sz:15 ][ addr:1042 sz:3 ][ addr:1045 sz:1 ][ addr:1046 sz:54 ]

Free(ptr[5])
returned 0
Free List [ Size 6 ]: [ addr:1004 sz:4 ][ addr:1008 sz:15 ][ addr:1023 sz:19 ][ addr:1042 sz:3 ][ addr:1045 sz:1 ][ addr:1046 sz:54 ]
```

```
Free(ptr[729])
returned 0
Free List [ Size 31 ]: [ addr:1000 sz:4 ][ addr:1004 sz:4 ][ addr:1008 sz:8 ][ addr:1018 sz:5 ][ addr:1023 sz:9 ][ addr:1032 sz:1 ][ addr:1033 sz:1 ][ addr:1034 sz:1 ][ addr:1035 sz:5 ][ addr:1040 sz:1 ]
[ addr:1041 sz:1 ][ addr:1042 sz:3 ][ addr:1045 sz:1 ][ addr:1046 sz:10 ][ addr:1056 sz:1 ][ addr:1057 sz:1 ][ addr:1058 sz:1 ][ addr:1059 sz:2 ][ addr:1061 sz:1 ][ addr:1062 sz:2 ][ addr:1064 sz:7 ][ ad
dr:1071 sz:1 ][ addr:1072 sz:10 ][ addr:1082 sz:1 ][ addr:1083 sz:1 ][ addr:1084 sz:1 ][ addr:1085 sz:1 ][ addr:1086 sz:4 ][ addr:1090 sz:6 ][ addr:1096 sz:1 ][ addr:1097 sz:3 ]

Free(ptr[728])
returned 0
Free List [ Size 32 ]: [ addr:1000 sz:4 ][ addr:1004 sz:4 ][ addr:1008 sz:8 ][ addr:1016 sz:2 ][ addr:1018 sz:5 ][ addr:1023 sz:9 ][ addr:1032 sz:1 ][ addr:1033 sz:1 ][ addr:1034 sz:1 ][ addr:1035 sz:5 ]
[ addr:1040 sz:1 ][ addr:1041 sz:1 ][ addr:1042 sz:3 ][ addr:1045 sz:1 ][ addr:1046 sz:10 ][ addr:1056 sz:1 ][ addr:1057 sz:1 ][ addr:1058 sz:1 ][ addr:1059 sz:2 ][ addr:1061 sz:1 ][ addr:1062 sz:2 ][ ad
dr:1064 sz:7 ][ addr:1071 sz:1 ][ addr:1072 sz:10 ][ addr:1082 sz:1 ][ addr:1083 sz:1 ][ addr:1084 sz:1 ][ addr:1085 sz:1 ][ addr:1086 sz:4 ][ addr:1090 sz:6 ][ addr:1096 sz:1 ][ addr:1097 sz:3 ]

ptr[730] = Alloc(5) returned 1018 (searched 32 elements)
Free List [ Size 31 ]: [ addr:1000 sz:4 ][ addr:1004 sz:4 ][ addr:1008 sz:8 ][ addr:1016 sz:2 ][ addr:1023 sz:9 ][ addr:1032 sz:1 ][ addr:1033 sz:1 ][ addr:1034 sz:1 ][ addr:1035 sz:5 ][ addr:1040 sz:1 ]
[ addr:1041 sz:1 ][ addr:1042 sz:3 ][ addr:1045 sz:1 ][ addr:1046 sz:10 ][ addr:1056 sz:1 ][ addr:1057 sz:1 ][ addr:1058 sz:1 ][ addr:1059 sz:2 ][ addr:1061 sz:1 ][ addr:1062 sz:2 ][ addr:1064 sz:7 ][ ad
dr:1071 sz:1 ][ addr:1072 sz:10 ][ addr:1082 sz:1 ][ addr:1083 sz:1 ][ addr:1084 sz:1 ][ addr:1085 sz:1 ][ addr:1086 sz:4 ][ addr:1090 sz:6 ][ addr:1096 sz:1 ][ addr:1097 sz:3 ]

Free(ptr[730])
returned 0
Free List [ Size 32 ]: [ addr:1000 sz:4 ][ addr:1004 sz:4 ][ addr:1008 sz:8 ][ addr:1016 sz:2 ][ addr:1018 sz:5 ][ addr:1023 sz:9 ][ addr:1032 sz:1 ][ addr:1033 sz:1 ][ addr:1034 sz:1 ][ addr:1035 sz:5 ]
[ addr:1040 sz:1 ][ addr:1041 sz:1 ][ addr:1042 sz:3 ][ addr:1045 sz:1 ][ addr:1046 sz:10 ][ addr:1056 sz:1 ][ addr:1057 sz:1 ][ addr:1058 sz:1 ][ addr:1059 sz:2 ][ addr:1061 sz:1 ][ addr:1062 sz:2 ][ ad
dr:1064 sz:7 ][ addr:1071 sz:1 ][ addr:1072 sz:10 ][ addr:1082 sz:1 ][ addr:1083 sz:1 ][ addr:1084 sz:1 ][ addr:1085 sz:1 ][ addr:1086 sz:4 ][ addr:1090 sz:6 ][ addr:1096 sz:1 ][ addr:1097 sz:3 ]

ptr[731] = Alloc(29) returned -1 (searched 32 elements)
Free List [ Size 32 ]: [ addr:1000 sz:4 ][ addr:1004 sz:4 ][ addr:1008 sz:8 ][ addr:1016 sz:2 ][ addr:1018 sz:5 ][ addr:1023 sz:9 ][ addr:1032 sz:1 ][ addr:1033 sz:1 ][ addr:1034 sz:1 ][ addr:1035 sz:5 ]
[ addr:1040 sz:1 ][ addr:1041 sz:1 ][ addr:1042 sz:3 ][ addr:1045 sz:1 ][ addr:1046 sz:10 ][ addr:1056 sz:1 ][ addr:1057 sz:1 ][ addr:1058 sz:1 ][ addr:1059 sz:2 ][ addr:1061 sz:1 ][ addr:1062 sz:2 ][ ad
dr:1064 sz:7 ][ addr:1071 sz:1 ][ addr:1072 sz:10 ][ addr:1082 sz:1 ][ addr:1083 sz:1 ][ addr:1084 sz:1 ][ addr:1085 sz:1 ][ addr:1086 sz:4 ][ addr:1090 sz:6 ][ addr:1096 sz:1 ][ addr:1097 sz:3 ]

ptr[732] = Alloc(28) returned -1 (searched 32 elements)
Free List [ Size 32 ]: [ addr:1000 sz:4 ][ addr:1004 sz:4 ][ addr:1008 sz:8 ][ addr:1016 sz:2 ][ addr:1018 sz:5 ][ addr:1023 sz:9 ][ addr:1032 sz:1 ][ addr:1033 sz:1 ][ addr:1034 sz:1 ][ addr:1035 sz:5 ]
[ addr:1040 sz:1 ][ addr:1041 sz:1 ][ addr:1042 sz:3 ][ addr:1045 sz:1 ][ addr:1046 sz:10 ][ addr:1056 sz:1 ][ addr:1057 sz:1 ][ addr:1058 sz:1 ][ addr:1059 sz:2 ][ addr:1061 sz:1 ][ addr:1062 sz:2 ][ ad
dr:1064 sz:7 ][ addr:1071 sz:1 ][ addr:1072 sz:10 ][ addr:1082 sz:1 ][ addr:1083 sz:1 ][ addr:1084 sz:1 ][ addr:1085 sz:1 ][ addr:1086 sz:4 ][ addr:1090 sz:6 ][ addr:1096 sz:1 ][ addr:1097 sz:3 ]

ptr[733] = Alloc(15) returned -1 (searched 32 elements)
Free List [ Size 32 ]: [ addr:1000 sz:4 ][ addr:1004 sz:4 ][ addr:1008 sz:8 ][ addr:1016 sz:2 ][ addr:1018 sz:5 ][ addr:1023 sz:9 ][ addr:1032 sz:1 ][ addr:1033 sz:1 ][ addr:1034 sz:1 ][ addr:1035 sz:5 ]
[ addr:1040 sz:1 ][ addr:1041 sz:1 ][ addr:1042 sz:3 ][ addr:1045 sz:1 ][ addr:1046 sz:10 ][ addr:1056 sz:1 ][ addr:1057 sz:1 ][ addr:1058 sz:1 ][ addr:1059 sz:2 ][ addr:1061 sz:1 ][ addr:1062 sz:2 ][ ad
dr:1064 sz:7 ][ addr:1071 sz:1 ][ addr:1072 sz:10 ][ addr:1082 sz:1 ][ addr:1083 sz:1 ][ addr:1084 sz:1 ][ addr:1085 sz:1 ][ addr:1086 sz:4 ][ addr:1090 sz:6 ][ addr:1096 sz:1 ][ addr:1097 sz:3 ]

ptr[734] = Alloc(12) returned -1 (searched 32 elements)
Free List [ Size 32 ]: [ addr:1000 sz:4 ][ addr:1004 sz:4 ][ addr:1008 sz:8 ][ addr:1016 sz:2 ][ addr:1018 sz:5 ][ addr:1023 sz:9 ][ addr:1032 sz:1 ][ addr:1033 sz:1 ][ addr:1034 sz:1 ][ addr:1035 sz:5 ]
[ addr:1040 sz:1 ][ addr:1041 sz:1 ][ addr:1042 sz:3 ][ addr:1045 sz:1 ][ addr:1046 sz:10 ][ addr:1056 sz:1 ][ addr:1057 sz:1 ][ addr:1058 sz:1 ][ addr:1059 sz:2 ][ addr:1061 sz:1 ][ addr:1062 sz:2 ][ ad
dr:1064 sz:7 ][ addr:1071 sz:1 ][ addr:1072 sz:10 ][ addr:1082 sz:1 ][ addr:1083 sz:1 ][ addr:1084 sz:1 ][ addr:1085 sz:1 ][ addr:1086 sz:4 ][ addr:1090 sz:6 ][ addr:1096 sz:1 ][ addr:1097 sz:3 ]

ptr[735] = Alloc(8) returned 1008 (searched 32 elements)
Free List [ Size 31 ]: [ addr:1000 sz:4 ][ addr:1004 sz:4 ][ addr:1016 sz:2 ][ addr:1018 sz:5 ][ addr:1023 sz:9 ][ addr:1032 sz:1 ][ addr:1033 sz:1 ][ addr:1034 sz:1 ][ addr:1035 sz:5 ][ addr:1040 sz:1 ]
[ addr:1041 sz:1 ][ addr:1042 sz:3 ][ addr:1045 sz:1 ][ addr:1046 sz:10 ][ addr:1056 sz:1 ][ addr:1057 sz:1 ][ addr:1058 sz:1 ][ addr:1059 sz:2 ][ addr:1061 sz:1 ][ addr:1062 sz:2 ][ addr:1064 sz:7 ][ ad
dr:1071 sz:1 ][ addr:1072 sz:10 ][ addr:1082 sz:1 ][ addr:1083 sz:1 ][ addr:1084 sz:1 ][ addr:1085 sz:1 ][ addr:1086 sz:4 ][ addr:1090 sz:6 ][ addr:1096 sz:1 ][ addr:1097 sz:3 ]

ptr[736] = Alloc(26) returned -1 (searched 31 elements)
Free List [ Size 31 ]: [ addr:1000 sz:4 ][ addr:1004 sz:4 ][ addr:1016 sz:2 ][ addr:1018 sz:5 ][ addr:1023 sz:9 ][ addr:1032 sz:1 ][ addr:1033 sz:1 ][ addr:1034 sz:1 ][ addr:1035 sz:5 ][ addr:1040 sz:1 ]
[ addr:1041 sz:1 ][ addr:1042 sz:3 ][ addr:1045 sz:1 ][ addr:1046 sz:10 ][ addr:1056 sz:1 ][ addr:1057 sz:1 ][ addr:1058 sz:1 ][ addr:1059 sz:2 ][ addr:1061 sz:1 ][ addr:1062 sz:2 ][ addr:1064 sz:7 ][ ad
dr:1071 sz:1 ][ addr:1072 sz:10 ][ addr:1082 sz:1 ][ addr:1083 sz:1 ][ addr:1084 sz:1 ][ addr:1085 sz:1 ][ addr:1086 sz:4 ][ addr:1090 sz:6 ][ addr:1096 sz:1 ][ addr:1097 sz:3 ]
```

```
malloc.py: error: no such option: --
onur@onur:~/Desktop$  python3 ./malloc.py -n 1000 -r 30 -c -C
seed 0
size 100
baseAddr 1000
headerSize 0
alignment -1
policy BEST
listOrder ADDRSORT
coalesce True
numOps 1000
range 30
percentAlloc 50
allocList
compute True

ptr[0] = Alloc(8) returned 1000 (searched 1 elements)
Free List [ Size 1 ]: [ addr:1008 sz:92 ]

Free(ptr[0])
returned 0
Free List [ Size 1 ]: [ addr:1000 sz:100 ]

ptr[1] = Alloc(15) returned 1000 (searched 1 elements)
Free List [ Size 1 ]: [ addr:1015 sz:85 ]

Free(ptr[1])
returned 0
Free List [ Size 1 ]: [ addr:1000 sz:100 ]

ptr[2] = Alloc(23) returned 1000 (searched 1 elements)
Free List [ Size 1 ]: [ addr:1023 sz:77 ]

Free(ptr[2])
returned 0
Free List [ Size 1 ]: [ addr:1000 sz:100 ]

ptr[3] = Alloc(22) returned 1000 (searched 1 elements)
Free List [ Size 1 ]: [ addr:1022 sz:78 ]

Free(ptr[3])
returned 0
Free List [ Size 1 ]: [ addr:1000 sz:100 ]

ptr[4] = Alloc(4) returned 1000 (searched 1 elements)
Free List [ Size 1 ]: [ addr:1004 sz:96 ]

ptr[5] = Alloc(19) returned 1004 (searched 1 elements)
Free List [ Size 1 ]: [ addr:1023 sz:77 ]

Free(ptr[5])
returned 0
```

```
Free(ptr[534])
returned 0
Free List [ Size 2 ]: [ addr:1000 sz:69 ][ addr:1094 sz:6 ]

ptr[535] = Alloc(25) returned 1000 (searched 2 elements)
Free List [ Size 2 ]: [ addr:1025 sz:44 ][ addr:1094 sz:6 ]

ptr[536] = Alloc(12) returned 1025 (searched 2 elements)
Free List [ Size 2 ]: [ addr:1037 sz:32 ][ addr:1094 sz:6 ]

Free(ptr[536])
returned 0
Free List [ Size 2 ]: [ addr:1025 sz:44 ][ addr:1094 sz:6 ]

ptr[537] = Alloc(13) returned 1025 (searched 2 elements)
Free List [ Size 2 ]: [ addr:1038 sz:31 ][ addr:1094 sz:6 ]

Free(ptr[532])
returned 0
Free List [ Size 2 ]: [ addr:1038 sz:55 ][ addr:1094 sz:6 ]

Free(ptr[535])
returned 0
Free List [ Size 3 ]: [ addr:1000 sz:25 ][ addr:1038 sz:55 ][ addr:1094 sz:6 ]

Free(ptr[537])
returned 0
Free List [ Size 2 ]: [ addr:1000 sz:93 ][ addr:1094 sz:6 ]

ptr[538] = Alloc(27) returned 1000 (searched 2 elements)
Free List [ Size 2 ]: [ addr:1027 sz:66 ][ addr:1094 sz:6 ]

Free(ptr[538])
returned 0
Free List [ Size 2 ]: [ addr:1000 sz:93 ][ addr:1094 sz:6 ]

ptr[539] = Alloc(26) returned 1000 (searched 2 elements)
Free List [ Size 2 ]: [ addr:1026 sz:67 ][ addr:1094 sz:6 ]

Free(ptr[533])
returned 0
Free List [ Size 1 ]: [ addr:1026 sz:74 ]

Free(ptr[539])
returned 0
Free List [ Size 1 ]: [ addr:1000 sz:100 ]

ptr[540] = Alloc(29) returned 1000 (searched 1 elements)
Free List [ Size 1 ]: [ addr:1029 sz:71 ]

Free(ptr[540])
returned 0
Free List [ Size 1 ]: [ addr:1000 sz:100 ]
```

6- Ayrılan kesir -P yüzdesini 50'den yüksek olarak değiştirdiğinizde ne olur? 100'e yaklaştıkça tahsislere ne olur? Yüzde 0'a yaklaşırkenne olur ?

>Tahsis talimatının yüzdesi 50 den büyükse tahsis işlemi elbet gerçekleşir.Tahsis talebinin yüzdesi 50'nin altına düşsede simulator hala yüzde 50 tahsis talebini üretir.Çünkü tahsis edilmemiş hafıza boşaltılamaz.

Tahsis talimatının yüzdesi yüksek olduğunda,tahsis işleminin başarılı olma olasılığı düşüktür.

Tahsis talimatının yüzdesi düşük olduğunda,tahsis işleminin başarılı olma olasılığı ise yüksektir.

```
onur@onur:~/Desktop$ S pyhton3 ./malloc.py -c -n 1000 -P 1
S: command not found
onur@onur:~/Desktop$ python3 ./malloc.py -c -n 1000 -P 1
seed 0
size 100
baseAddr 1000
headerSize 0
alignment -1
policy BEST
listOrder ADDRSORT
coalesce False
numOps 1000
range 10
percentAlloc 1
allocList
compute True

ptr[0] = Alloc(5) returned 1000 (searched 1 elements)
Free List [ Size 1 ]: [ addr:1005 sz:95 ]

Free(ptr[0])
returned 0
Free List [ Size 2 ]: [ addr:1000 sz:5 ][ addr:1005 sz:95 ]

ptr[1] = Alloc(2) returned 1000 (searched 2 elements)
Free List [ Size 2 ]: [ addr:1002 sz:3 ][ addr:1005 sz:95 ]

Free(ptr[1])
returned 0
Free List [ Size 3 ]: [ addr:1000 sz:2 ][ addr:1002 sz:3 ][ addr:1005 sz:95 ]

ptr[2] = Alloc(9) returned 1005 (searched 3 elements)
Free List [ Size 3 ]: [ addr:1000 sz:2 ][ addr:1002 sz:3 ][ addr:1014 sz:86 ]

Free(ptr[2])
returned 0
Free List [ Size 4 ]: [ addr:1000 sz:2 ][ addr:1002 sz:3 ][ addr:1005 sz:9 ][ addr:1014 sz:86 ]

ptr[3] = Alloc(2) returned 1000 (searched 4 elements)
Free List [ Size 3 ]: [ addr:1002 sz:3 ][ addr:1005 sz:9 ][ addr:1014 sz:86 ]

Free(ptr[3])
returned 0
Free List [ Size 4 ]: [ addr:1000 sz:2 ][ addr:1002 sz:3 ][ addr:1005 sz:9 ][ addr:1014 sz:86 ]

ptr[4] = Alloc(4) returned 1005 (searched 4 elements)
Free List [ Size 4 ]: [ addr:1000 sz:2 ][ addr:1002 sz:3 ][ addr:1009 sz:5 ][ addr:1014 sz:86 ]

Free(ptr[4])
returned 0
Free List [ Size 5 ]: [ addr:1000 sz:2 ][ addr:1002 sz:3 ][ addr:1005 sz:4 ][ addr:1009 sz:5 ][ addr:1014 sz:86 ]
```

```
Free List [ Size 22 ]: [ addr:1000 sz:1 ][ addr:1001 sz:1 ][ addr:1002 sz:2 ][ addr:1004 sz:1 ][ addr:1005 sz:4 ][ addr:1009 sz:5 ][ addr:1014 sz:6 ][ addr:1020 sz:1 ][ addr:1021 sz:3 ][ addr:1024 sz:7 ]
[ addr:1031 sz:2 ][ addr:1033 sz:7 ][ addr:1040 sz:1 ][ addr:1041 sz:1 ][ addr:1042 sz:1 ][ addr:1043 sz:8 ][ addr:1051 sz:1 ][ addr:1052 sz:1 ][ addr:1053 sz:9 ][ addr:1062 sz:1 ][ addr:1063 sz:10 ][ ad
dr:1073 sz:27 ]

ptr[495] = Alloc(10) returned 1063 (searched 22 elements)
Free List [ Size 21 ]: [ addr:1000 sz:1 ][ addr:1001 sz:1 ][ addr:1002 sz:2 ][ addr:1004 sz:1 ][ addr:1005 sz:4 ][ addr:1009 sz:5 ][ addr:1014 sz:6 ][ addr:1020 sz:1 ][ addr:1021 sz:3 ][ addr:1024 sz:7 ]
[ addr:1031 sz:2 ][ addr:1033 sz:7 ][ addr:1040 sz:1 ][ addr:1041 sz:1 ][ addr:1042 sz:1 ][ addr:1043 sz:8 ][ addr:1051 sz:1 ][ addr:1052 sz:1 ][ addr:1053 sz:9 ][ addr:1062 sz:1 ][ addr:1073 sz:27 ]

Free(ptr[495])
returned 0
Free List [ Size 22 ]: [ addr:1000 sz:1 ][ addr:1001 sz:1 ][ addr:1002 sz:2 ][ addr:1004 sz:1 ][ addr:1005 sz:4 ][ addr:1009 sz:5 ][ addr:1014 sz:6 ][ addr:1020 sz:1 ][ addr:1021 sz:3 ][ addr:1024 sz:7 ]
[ addr:1031 sz:2 ][ addr:1033 sz:7 ][ addr:1040 sz:1 ][ addr:1041 sz:1 ][ addr:1042 sz:1 ][ addr:1043 sz:8 ][ addr:1051 sz:1 ][ addr:1052 sz:1 ][ addr:1053 sz:9 ][ addr:1062 sz:1 ][ addr:1063 sz:10 ][ ad
dr:1073 sz:27 ]

ptr[496] = Alloc(10) returned 1063 (searched 22 elements)
Free List [ Size 21 ]: [ addr:1000 sz:1 ][ addr:1001 sz:1 ][ addr:1002 sz:2 ][ addr:1004 sz:1 ][ addr:1005 sz:4 ][ addr:1009 sz:5 ][ addr:1014 sz:6 ][ addr:1020 sz:1 ][ addr:1021 sz:3 ][ addr:1024 sz:7 ]
[ addr:1031 sz:2 ][ addr:1033 sz:7 ][ addr:1040 sz:1 ][ addr:1041 sz:1 ][ addr:1042 sz:1 ][ addr:1043 sz:8 ][ addr:1051 sz:1 ][ addr:1052 sz:1 ][ addr:1053 sz:9 ][ addr:1062 sz:1 ][ addr:1073 sz:27 ]

Free(ptr[496])
returned 0
Free List [ Size 22 ]: [ addr:1000 sz:1 ][ addr:1001 sz:1 ][ addr:1002 sz:2 ][ addr:1004 sz:1 ][ addr:1005 sz:4 ][ addr:1009 sz:5 ][ addr:1014 sz:6 ][ addr:1020 sz:1 ][ addr:1021 sz:3 ][ addr:1024 sz:7 ]
[ addr:1031 sz:2 ][ addr:1033 sz:7 ][ addr:1040 sz:1 ][ addr:1041 sz:1 ][ addr:1042 sz:1 ][ addr:1043 sz:8 ][ addr:1051 sz:1 ][ addr:1052 sz:1 ][ addr:1053 sz:9 ][ addr:1062 sz:1 ][ addr:1063 sz:10 ][ ad
dr:1073 sz:27 ]

ptr[497] = Alloc(1) returned 1000 (searched 22 elements)
Free List [ Size 21 ]: [ addr:1001 sz:1 ][ addr:1002 sz:2 ][ addr:1004 sz:1 ][ addr:1005 sz:4 ][ addr:1009 sz:5 ][ addr:1014 sz:6 ][ addr:1020 sz:1 ][ addr:1021 sz:3 ][ addr:1024 sz:7 ][ addr:1031 sz:2 ]
[ addr:1033 sz:7 ][ addr:1040 sz:1 ][ addr:1041 sz:1 ][ addr:1042 sz:1 ][ addr:1043 sz:8 ][ addr:1051 sz:1 ][ addr:1052 sz:1 ][ addr:1053 sz:9 ][ addr:1062 sz:1 ][ addr:1063 sz:10 ][ addr:1073 sz:27 ]

Free(ptr[497])
returned 0
Free List [ Size 22 ]: [ addr:1000 sz:1 ][ addr:1001 sz:1 ][ addr:1002 sz:2 ][ addr:1004 sz:1 ][ addr:1005 sz:4 ][ addr:1009 sz:5 ][ addr:1014 sz:6 ][ addr:1020 sz:1 ][ addr:1021 sz:3 ][ addr:1024 sz:7 ]
[ addr:1031 sz:2 ][ addr:1033 sz:7 ][ addr:1040 sz:1 ][ addr:1041 sz:1 ][ addr:1042 sz:1 ][ addr:1043 sz:8 ][ addr:1051 sz:1 ][ addr:1052 sz:1 ][ addr:1053 sz:9 ][ addr:1062 sz:1 ][ addr:1063 sz:10 ][ ad
dr:1073 sz:27 ]

ptr[498] = Alloc(2) returned 1002 (searched 22 elements)
Free List [ Size 21 ]: [ addr:1000 sz:1 ][ addr:1001 sz:1 ][ addr:1004 sz:1 ][ addr:1005 sz:4 ][ addr:1009 sz:5 ][ addr:1014 sz:6 ][ addr:1020 sz:1 ][ addr:1021 sz:3 ][ addr:1024 sz:7 ][ addr:1031 sz:2 ]
[ addr:1033 sz:7 ][ addr:1040 sz:1 ][ addr:1041 sz:1 ][ addr:1042 sz:1 ][ addr:1043 sz:8 ][ addr:1051 sz:1 ][ addr:1052 sz:1 ][ addr:1053 sz:9 ][ addr:1062 sz:1 ][ addr:1063 sz:10 ][ addr:1073 sz:27 ]

Free(ptr[498])
returned 0
Free List [ Size 22 ]: [ addr:1000 sz:1 ][ addr:1001 sz:1 ][ addr:1002 sz:2 ][ addr:1004 sz:1 ][ addr:1005 sz:4 ][ addr:1009 sz:5 ][ addr:1014 sz:6 ][ addr:1020 sz:1 ][ addr:1021 sz:3 ][ addr:1024 sz:7 ]
[ addr:1031 sz:2 ][ addr:1033 sz:7 ][ addr:1040 sz:1 ][ addr:1041 sz:1 ][ addr:1042 sz:1 ][ addr:1043 sz:8 ][ addr:1051 sz:1 ][ addr:1052 sz:1 ][ addr:1053 sz:9 ][ addr:1062 sz:1 ][ addr:1063 sz:10 ][ ad
dr:1073 sz:27 ]

ptr[499] = Alloc(5) returned 1009 (searched 22 elements)
Free List [ Size 21 ]: [ addr:1000 sz:1 ][ addr:1001 sz:1 ][ addr:1002 sz:2 ][ addr:1004 sz:1 ][ addr:1005 sz:4 ][ addr:1014 sz:6 ][ addr:1020 sz:1 ][ addr:1021 sz:3 ][ addr:1024 sz:7 ][ addr:1031 sz:2 ]
[ addr:1033 sz:7 ][ addr:1040 sz:1 ][ addr:1041 sz:1 ][ addr:1042 sz:1 ][ addr:1043 sz:8 ][ addr:1051 sz:1 ][ addr:1052 sz:1 ][ addr:1053 sz:9 ][ addr:1062 sz:1 ][ addr:1063 sz:10 ][ addr:1073 sz:27 ]

Free(ptr[499])
returned 0
Free List [ Size 22 ]: [ addr:1000 sz:1 ][ addr:1001 sz:1 ][ addr:1002 sz:2 ][ addr:1004 sz:1 ][ addr:1005 sz:4 ][ addr:1009 sz:5 ][ addr:1014 sz:6 ][ addr:1020 sz:1 ][ addr:1021 sz:3 ][ addr:1024 sz:7 ]
[ addr:1031 sz:2 ][ addr:1033 sz:7 ][ addr:1040 sz:1 ][ addr:1041 sz:1 ][ addr:1042 sz:1 ][ addr:1043 sz:8 ][ addr:1051 sz:1 ][ addr:1052 sz:1 ][ addr:1053 sz:9 ][ addr:1062 sz:1 ][ addr:1063 sz:10 ][ ad
dr:1073 sz:27 ]
```

```
onur@onur:~/Desktop$ S pyhton3 ./malloc.py -c -n 1000 -P 1
S: command not found
onur@onur:~/Desktop$ python3 ./malloc.py -c -n 1000 -P 1
seed 0
size 100
baseAddr 1000
headerSize 0
alignment -1
policy BEST
listOrder ADDRSORT
coalesce False
numOps 1000
range 10
percentAlloc 1
allocList
compute True

ptr[0] = Alloc(5) returned 1000 (searched 1 elements)
Free List [ Size 1 ]: [ addr:1005 sz:95 ]

Free(ptr[0])
returned 0
Free List [ Size 2 ]: [ addr:1000 sz:5 ][ addr:1005 sz:95 ]

ptr[1] = Alloc(2) returned 1000 (searched 2 elements)
Free List [ Size 2 ]: [ addr:1002 sz:3 ][ addr:1005 sz:95 ]

Free(ptr[1])
returned 0
Free List [ Size 3 ]: [ addr:1000 sz:2 ][ addr:1002 sz:3 ][ addr:1005 sz:95 ]

ptr[2] = Alloc(9) returned 1005 (searched 3 elements)
Free List [ Size 3 ]: [ addr:1000 sz:2 ][ addr:1002 sz:3 ][ addr:1014 sz:86 ]

Free(ptr[2])
returned 0
Free List [ Size 4 ]: [ addr:1000 sz:2 ][ addr:1002 sz:3 ][ addr:1005 sz:9 ][ addr:1014 sz:86 ]

ptr[3] = Alloc(2) returned 1000 (searched 4 elements)
Free List [ Size 3 ]: [ addr:1002 sz:3 ][ addr:1005 sz:9 ][ addr:1014 sz:86 ]

Free(ptr[3])
returned 0
Free List [ Size 4 ]: [ addr:1000 sz:2 ][ addr:1002 sz:3 ][ addr:1005 sz:9 ][ addr:1014 sz:86 ]

ptr[4] = Alloc(4) returned 1005 (searched 4 elements)
Free List [ Size 4 ]: [ addr:1000 sz:2 ][ addr:1002 sz:3 ][ addr:1009 sz:5 ][ addr:1014 sz:86 ]

Free(ptr[4])
returned 0
Free List [ Size 5 ]: [ addr:1000 sz:2 ][ addr:1002 sz:3 ][ addr:1005 sz:4 ][ addr:1009 sz:5 ][ addr:1014 sz:86 ]
```

```
onur@onur:~/Desktop$ S pyhton3 ./malloc.py -c -n 1000 -P 1
S: command not found
onur@onur:~/Desktop$ python3 ./malloc.py -c -n 1000 -P 1
seed 0
size 100
baseAddr 1000
headerSize 0
alignment -1
policy BEST
listOrder ADDRSORT
coalesce False
numOps 1000
range 10
percentAlloc 1
allocList
compute True

ptr[0] = Alloc(5) returned 1000 (searched 1 elements)
Free List [ Size 1 ]: [ addr:1005 sz:95 ]

Free(ptr[0])
returned 0
Free List [ Size 2 ]: [ addr:1000 sz:5 ][ addr:1005 sz:95 ]

ptr[1] = Alloc(2) returned 1000 (searched 2 elements)
Free List [ Size 2 ]: [ addr:1002 sz:3 ][ addr:1005 sz:95 ]

Free(ptr[1])
returned 0
Free List [ Size 3 ]: [ addr:1000 sz:2 ][ addr:1002 sz:3 ][ addr:1005 sz:95 ]

ptr[2] = Alloc(9) returned 1005 (searched 3 elements)
Free List [ Size 3 ]: [ addr:1000 sz:2 ][ addr:1002 sz:3 ][ addr:1014 sz:86 ]

Free(ptr[2])
returned 0
Free List [ Size 4 ]: [ addr:1000 sz:2 ][ addr:1002 sz:3 ][ addr:1005 sz:9 ][ addr:1014 sz:86 ]

ptr[3] = Alloc(2) returned 1000 (searched 4 elements)
Free List [ Size 3 ]: [ addr:1002 sz:3 ][ addr:1005 sz:9 ][ addr:1014 sz:86 ]

Free(ptr[3])
returned 0
Free List [ Size 4 ]: [ addr:1000 sz:2 ][ addr:1002 sz:3 ][ addr:1005 sz:9 ][ addr:1014 sz:86 ]

ptr[4] = Alloc(4) returned 1005 (searched 4 elements)
Free List [ Size 4 ]: [ addr:1000 sz:2 ][ addr:1002 sz:3 ][ addr:1009 sz:5 ][ addr:1014 sz:86 ]

Free(ptr[4])
returned 0
Free List [ Size 5 ]: [ addr:1000 sz:2 ][ addr:1002 sz:3 ][ addr:1005 sz:4 ][ addr:1009 sz:5 ][ addr:1014 sz:86 ]
```

7- Son derece parçalı bir boş alan oluşturmak için ne tür özel isteklerde bulunabilirsiniz? Parçalı serbest listeler oluşturmak için -A bayrağını kullanın ve farklı ilkelerin ve seçeneklerin serbest listenin organizasyonunu nasıl değiştirdiğini görün.

>Adres sıralama ve birleştirme işlemleri gerçekleşirken.Parçalı boş bir alan oluşturmak mümkün değildir.

```
onur@onur: $ python3 ./malloc.py -c -A +10,+15,+20,+25,+30,-0,-1,-2,-3,-4
seed 0
size 100
baseAddr 1000
headerSize 0
alignment -1
policy BEST
listOrder ADDRSORT
coalesce False
numOps 10
range 10
percentAlloc 50
allocList +10,+15,+20,+25,+30,-0,-1,-2,-3,-4
compute True

ptr[0] = Alloc(10) returned 1000 (searched 1 elements)
Free List [ Size 1 ]: [ addr:1010 sz:90 ]

ptr[1] = Alloc(15) returned 1010 (searched 1 elements)
Free List [ Size 1 ]: [ addr:1025 sz:75 ]

ptr[2] = Alloc(20) returned 1025 (searched 1 elements)
Free List [ Size 1 ]: [ addr:1045 sz:55 ]

ptr[3] = Alloc(25) returned 1045 (searched 1 elements)
Free List [ Size 1 ]: [ addr:1070 sz:30 ]

ptr[4] = Alloc(30) returned 1070 (searched 1 elements)
Free List [ Size 0 ]:

Free(ptr[0])
returned 0
Free List [ Size 1 ]: [ addr:1000 sz:10 ]

Free(ptr[1])
returned 0
Free List [ Size 2 ]: [ addr:1000 sz:10 ][ addr:1010 sz:15 ]

Free(ptr[2])
returned 0
Free List [ Size 3 ]: [ addr:1000 sz:10 ][ addr:1010 sz:15 ][ addr:1025 sz:20 ]

Free(ptr[3])
returned 0
Free List [ Size 4 ]: [ addr:1000 sz:10 ][ addr:1010 sz:15 ][ addr:1025 sz:20 ][ addr:1045 sz:25 ]

Free(ptr[4])
returned 0
Free List [ Size 5 ]: [ addr:1000 sz:10 ][ addr:1010 sz:15 ][ addr:1025 sz:20 ][ addr:1045 sz:25 ][ addr:1070 sz:30 ]
```

```
onur@onur:~$ python3 ./malloc.py -c -A +10,+15,+20,+25,+30,-0,-1,-2,-3,-4 -l SIZESORT+ -C
seed 0
size 100
baseAddr 1000
headerSize 0
alignment -1
policy BEST
listOrder SIZESORT+
coalesce True
numOps 10
range 10
percentAlloc 50
allocList +10,+15,+20,+25,+30,-0,-1,-2,-3,-4
compute True

ptr[0] = Alloc(10) returned 1000 (searched 1 elements)
Free List [ Size 1 ]: [ addr:1010 sz:90 ]

ptr[1] = Alloc(15) returned 1010 (searched 1 elements)
Free List [ Size 1 ]: [ addr:1025 sz:75 ]

ptr[2] = Alloc(20) returned 1025 (searched 1 elements)
Free List [ Size 1 ]: [ addr:1045 sz:55 ]

ptr[3] = Alloc(25) returned 1045 (searched 1 elements)
Free List [ Size 1 ]: [ addr:1070 sz:30 ]

ptr[4] = Alloc(30) returned 1070 (searched 1 elements)
Free List [ Size 0 ]:

Free(ptr[0])
returned 0
Free List [ Size 1 ]: [ addr:1000 sz:10 ]

Free(ptr[1])
returned 0
Free List [ Size 1 ]: [ addr:1000 sz:25 ]

Free(ptr[2])
returned 0
Free List [ Size 2 ]: [ addr:1025 sz:20 ][ addr:1000 sz:25 ]

Free(ptr[3])
returned 0
Free List [ Size 3 ]: [ addr:1025 sz:20 ][ addr:1000 sz:25 ][ addr:1045 sz:25 ]

Free(ptr[4])
returned 0
Free List [ Size 3 ]: [ addr:1025 sz:20 ][ addr:1000 sz:25 ][ addr:1045 sz:55 ]
```

```
onur@onur:~$ python3 ./malloc.py -c -A +10,+15,+20,+25,+30,-0,-1,-2,-3,-4 -l SIZESORT- -C
seed 0
size 100
baseAddr 1000
headerSize 0
alignment -1
policy BEST
listOrder SIZESORT-
coalesce True
numOps 10
range 10
percentAlloc 50
allocList +10,+15,+20,+25,+30,-0,-1,-2,-3,-4
compute True

ptr[0] = Alloc(10) returned 1000 (searched 1 elements)
Free List [ Size 1 ]: [ addr:1010 sz:90 ]

ptr[1] = Alloc(15) returned 1010 (searched 1 elements)
Free List [ Size 1 ]: [ addr:1025 sz:75 ]

ptr[2] = Alloc(20) returned 1025 (searched 1 elements)
Free List [ Size 1 ]: [ addr:1045 sz:55 ]

ptr[3] = Alloc(25) returned 1045 (searched 1 elements)
Free List [ Size 1 ]: [ addr:1070 sz:30 ]

ptr[4] = Alloc(30) returned 1070 (searched 1 elements)
Free List [ Size 0 ]:

Free(ptr[0])
returned 0
Free List [ Size 1 ]: [ addr:1000 sz:10 ]

Free(ptr[1])
returned 0
Free List [ Size 2 ]: [ addr:1010 sz:15 ][ addr:1000 sz:10 ]

Free(ptr[2])
returned 0
Free List [ Size 3 ]: [ addr:1025 sz:20 ][ addr:1010 sz:15 ][ addr:1000 sz:10 ]

Free(ptr[3])
returned 0
Free List [ Size 4 ]: [ addr:1045 sz:25 ][ addr:1025 sz:20 ][ addr:1010 sz:15 ][ addr:1000 sz:10 ]

Free(ptr[4])
returned 0
Free List [ Size 5 ]: [ addr:1070 sz:30 ][ addr:1045 sz:25 ][ addr:1025 sz:20 ][ addr:1010 sz:15 ][ addr:1000 sz:10 ]
```